Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Implementació d'una estació base LTE (down-link)

TITULACIÓ: Grau en Enginyeria de Sistemes de Telecomunicació

AUTOR: Gerard Romero Lorenzo

DIRECTOR: Antoni Gelonch Bosch

DATA: 22 de Juliol del 2015

Títol: Implementació d'una estació base LTE (down-link)

Autor: Gerard Romero Lorenzo

Director: Antoni Gelonch Bosch

Data: 22 de juliol del 2015

Resum

En l'actualitat les tecnologies sense fils són les formes de comunicació més utilitzades en tot el planeta. Gràcies a aquestes, les persones poden intercanviar-se informació a centenars de milers de quilòmetres amb unes velocitats de transferència força elevades. Això és possible gràcies a continues millores en les seves tecnologies .

En aquest projecte s'ha intentat implementar un estàndard de comunicacions que avui en dia s'està instaurant en els països més importants de tot el món. Aquest estàndard és el Long Term Evolution (LTE).

Primerament, s'explicarà com estan estructurades totes les dades que transporta (senyals i canals de la capa física). A més a més, es veurà les passes per tal d'implementar tant l'estació base emissora com l'usuari receptor.

Per tal de comprovar que el sistema funciona correctament es faran proves a través d'un canal gaussià i d'un canal real. Per cada prova es veurà el resultat i la seva interpretació.

I per últim, s'introduirà la implementació de LTE en la tecnologia Software-Defined Radio (SDR). Aquesta ens permet oblidar la part hardware per centrar els esforços en la millora i optimització de la part software.

Title: Implementació d'una estació base LTE (down-link)

Author: Gerard Romero Lorenzo

Director: Antoni Gelonch Bosch

Date: July 22 th 2014

Overview

Nowadays wireless technologies are the most widely used communication technologies around the world. Thanks to them, people, located far away, can exchange information even at high data transfers. This has been possible thanks to the continuous improvements, for more than two decades, in the involved technologies.

In this project, it has been tried to implement the wireless communication standard currently being deployed around the world. This standard is known as Long Term Evolution (LTE).

First of all, we will explain the structure of a radio frame, or how has been organised the data (signals and channels on the physical layer) transfers. Secondly, we will follow a hands-on approach to describe the steps to implement the downlink for both the eNodeB and the user terminal.

In order to check our system we have performed several test using AWGN and real channels. The results from these test provides some insights about the quality of the implementation.

Finally, a verified donwlink implementation will be introduced in Software-Defined Radio (SDR) technology with the objective to abstract the hardware issues and focus in the software improvement and optimization.

ÍNDEX

LLISTAT DE FIGURES

LLISTAT DE TAULES

INTRODUCCIÓ

Long-Term Evolution (LTE) és el darrer estàndard desplegar en el món de les tecnologies sense fils i és capaç d'assolir les velocitat més ràpides en la transmissió de dades. Aquesta tecnologia, basada en OFDMA en Downlink y SC-FDMA en el Uplink, pot arribar a enviar y transmetre informació d'usuari a taxes de 150Mbps en condicions.

Aquesta tecnologia juntament amb una implementació basada en software o SDR (Software Defiened Radio), permet desplegar xarxes de comunicacions de forma ràpida i eficient. SDR requereix dur a terme múltiples processos en temps real i per tant, la reducció del temps de processat d'un sistema LTE s'ha de basar en la paral·lelització de tasques.

En el nostre projecte introduirem aquestes dues tecnologies per implementar una estació Base (Downlink) per tal que hi hagi una comunicació fluida. A més a més s'implementarà un UE tal de rebre el senyal generat i seguir tots els processos per descodificar-lo de forma òptima.

Durant el projecte veurem gran part del sistema de LTE, explicarem tot el que té que a veure amb LTE (estructura del frame, espectre, Canals, etc…) i les implementarem gràcies a la llibreria OpenSource libLTE/srsLTE. Per altre banda, dissenyarem un sistema multi-core en tecnologia SDR capaç d'incorporar la llibreria srsLTE i fer-la funcionar per transmetre i rebre LTE.

CAPÍTOL 1. SOFTWARE DEFINED RADIO

1.1. Introducció a Software Defined Radio

Software Defined Radio és un sistema de radiocomunicació capaç d'implementar software a partir dels components hardware (filtres, mescladors, etc...). El Foro SDR amb la col·laboració del Institut d'Enginyers Elèctrics i Electrònics han treballat amb una definició que proporciona coherència i una visió clara de la tecnologia i dels seus beneficis, i és la següent:

"*Radio en la que part o la totalitat de les funcions de la capa física són definides per software*" [1]

D'acord amb la definició aportada, SDR és una tecnologia amb una arquitectura flexible i rentable capaç de portar les comunicacions via Radio més enllà i facilitar la transferència d'informació. Sense aquesta tecnologia, els dispositius radio basats en hardware limiten la funcionalitat i solament poden modificar-se mitjançant el canvi dels seus component físics, com per exemple, canviar un filtre, un mesclador, o fins i tot, moduladors físics. Això pot ser molt costós i poc eficient, mentre que amb la tecnologia software, els components físics queden a un costat i es modificarà el software del sistema mitjançant actualitzacions, cosa que redueix bastant els costos i augmenta l'eficiència del sistema.

1.2. Projectes FlexNets

FlexNets (Flexible Wireless Communication System and Networks) és una iniciativa d'investigació de codi obert que explora nous medis per aprofitar la flexibilitat de les comunicacions Radio.

Els projectes d'aquesta iniciativa fan que sigui més fàcil el disseny i desenvolupament de la comunicació via radio. Això és degut al fet que proporcionen una plataforma d'investigació pel Software Defined Radio i proporcionen les eines de software per tal de compartir recursos de Hardware i de Software i gestionar-los per maximitzar els beneficis de la flexibilitat del sistema i augmentar l'eficiència.

Un del projectes capaços de dur a terme el desenvolupament de la comunicació via radio és el projecte ALOE. Aquest software ens proporciona eines per treballar amb una plataforma de multiprocessadors.

1.3. ALOE

ALOE (Abstraction Layer and Operating Environment) és un software que actua com a pont entre un sistema operatiu o bases de dades i aplicacions, o MIDDLEWARE, que facilita el desplegament de codi en mutliplataforma.

1.3.1. Arquitectura i capes d'ALOE

Aquest software radio té quatre capes de processament, cadascuna d'elles es mostren en la imatge a continuació.

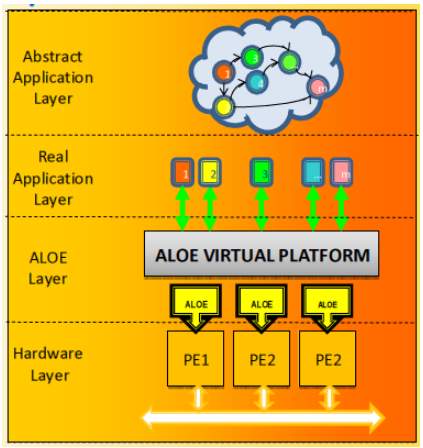


Fig 1. 1: Capes de processament ALOE [2]

La capa Hardware és un grup de Processadors (Processing Elements o PE) que estan interconnectades entre sí.

La capa ALOE és on el Software està introduït i funciona per separat en cada ordinador en que està connectat.

La capa d'aplicació és la capa on s'executa els waveforms programats. Està compost per tots els mòduls que s'han construït i totes les aplicacions que interconnecten els mòduls per tal de crear una cadena de processament.

I per últim, tenim la capa d'aplicació abstracte on tots els mòduls que s'han programat es gestionen i se'ls proporciona els recursos necessaris i/o disponibles.

Condicionats per l' implementació, ALOE utilitza alguns components i llibreries com les que es mostren en la següent imatges:

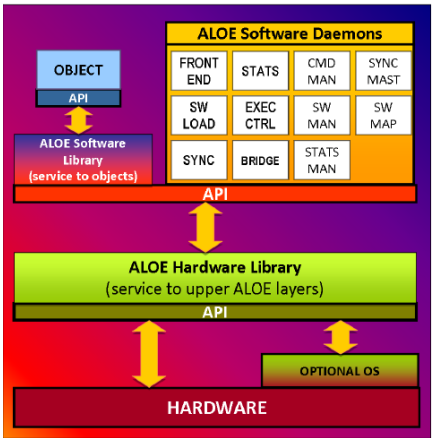


Fig 1. 2: Arquitectura d'ALOE [2]

A la capa API, ALOE utilitza les llibreries per utilitzar els objectes d'aquestes i així facilitar el processament del senyal, i a més utilitza el Software Daemons. Aquest Software és un grup de programes amb la finalitat de poder mantenir una correcte execució d'ALOE.

1.3.2. Plataforma Multiprocessament

ALOE té un gran avantatge respecte altres Softwares destintats a la comunicació radio i és que posseeix una plataforma de multiprocessament. Pot dividir les aplicacions en diferents processos (mòduls) i cadascun dels PE (Processors Elements) realitza un procés diferent de l'aplicació. Per tal de poder aconseguir un bon processat, els PE han d'estar sincronitzats en temps i han de processar les dades en un interval anomenat Time Slot (TS), és a dir, tots els mòduls de l'aplicació que s'estiguin executant s'han d'executar en un Time Slot.

Tot i tenir el gran avantatge del multiprocessament, ALOE posseeix un desavantatge important. Aquest desavantatge és la latència del senyal, és a dir, el temps total en passar tots els blocs de dades durant tot el processat fins arribar a la sortida.

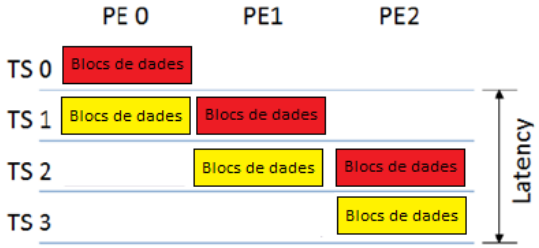


Fig 1. 3: Descripció gràfica multiprocessament

En una aplicació d'ALOE podem tenir diferents tipus de mòduls i no tots necessiten els mateixos per tal de processar la senyal. Tan és així, que el mateix software té un mecanisme de planificació, o Scheduling, del recursos disponibles, així pot donar més recursos a aquells mòduls que els necessitin.

1.3.3. Topologia

ALOE està distribuït en mòduls capaços d'interconnectar-se entre si per crear aplicacions. Tots els mòduls tenen la mateixa estructura, i està composta per una funció d'inicialització de les variables, una funció principal on se seleccionen les interfícies d'entrada i sortida i una funció d'aturada. Tots els mòduls tenen uns arxius de configuració que determinen el nombre d'interfícies d'entrada i de sortida i els tipus de dades que transporten, uns altres arxius per poder incloure les llibreries necessàries per a la seva execució, i arxius de configuracions de paràmetres interns.

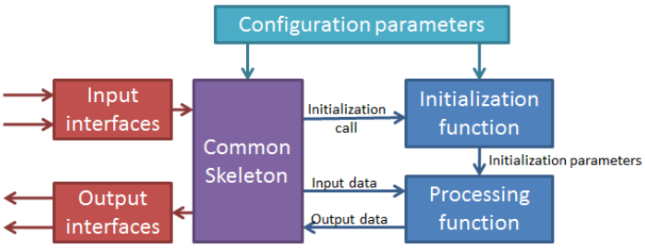


Fig 1. 4: Organització del mòduls d'ALOE

CAPÍTOL 2. CONCEPTES BÀSICS DE LTE

Long Term Evolution o LTE és un estàndard de comunicacions mòbils desenvolupat per 3GPP nascut per cobrir les necessitats dels usuaris que volien connexions de dades més ràpides tant de pujada com de baixada, per cobrir les necessitats dels fabricants i de les operadores que volien un estàndard menys complexa que el UMTS, per tal de reduir costos; i per últim, per assegurar la competitivitat del 3G en el futur.

La red LTE està formada per estacions base (eNodeB) i usuaris finals (UEs), entre d'altres. La informació que s'intercanvien entre si no és la mateixa en ambdós sentits perquè no tenen les mateixes característiques. Mentre que una eNodeB pot tenir amplificadors de potència de qualitat per evitar els problemes de linealitat i de PAPR, els UE no els poden tenir. Per aquest motiu l'eNodeB pot utilitzar una tecnologia basada en multiportadores anomenada OFDMA en l'enllaç descendent o DOWNLINK, mentre que els UE utilitzen una tecnologia de portadores simples anomenades SC-FDMA en l'enllaç ascendent o UPLINK.

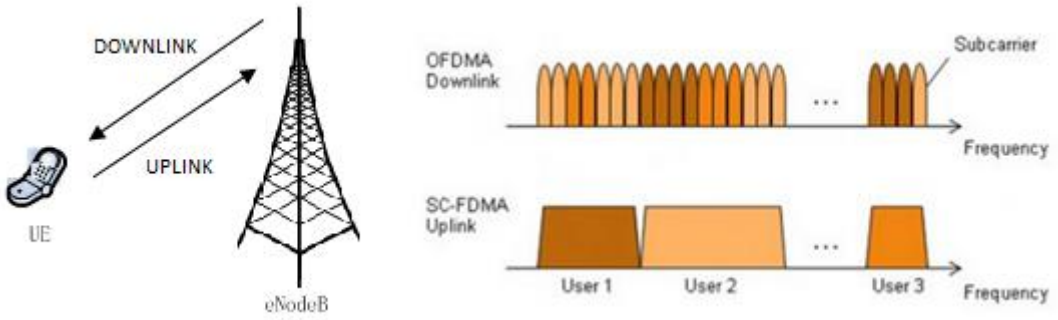


Fig 2. 1 Tecnologies en LTE

En els següents apartats es parlarà sobre com és l'estructura de les dades i s'explicaran tots els components de la capa física de LTE, tant els senyals de sincronisme com els canals físics que transporten la informació en l'enllaç DOWN-LINK.

2.1. Estructura LTE

El sistema de comunicació LTE té determinades dues estructures físiques per transmetre informació, aquestes estructures s'anomenen FDD i TDD. Ambdues estructures tenen com a punt en comú que estan determinades tant pel domini temporal com pel freqüencial, però el seu contingut no és el mateix.

En el cas del mode FDD utilitza estructures diferents pels enllaços DownLink i pel Uplink, mentre que en TDD la mateixa estructura pot enviar senyals d'enllaç Downlink com Uplink. Conceptes bàsics de LTE

S'explicar l'estructura FDD ja que serà la que s'implementarà a la part pràctica d'aquest projecte.

2.1.1. Estructura Frequency Division Duplex

L'Estructura FDD en el domini temporal està composta de Radio Frames amb una longitud temporal de 10ms cadascun. Cada Frame està dividit per deu subframes d'igual longitud i cadascun consta d'1ms. Cada subframes està compost de dos slots d'igual longitud i ocupen 0.5ms. Cadascun dels slots està compost de símbols OFDM, els quals el seu número pot variar depenent del tipus de prefix cíclic associat a l'estructura de la trama.

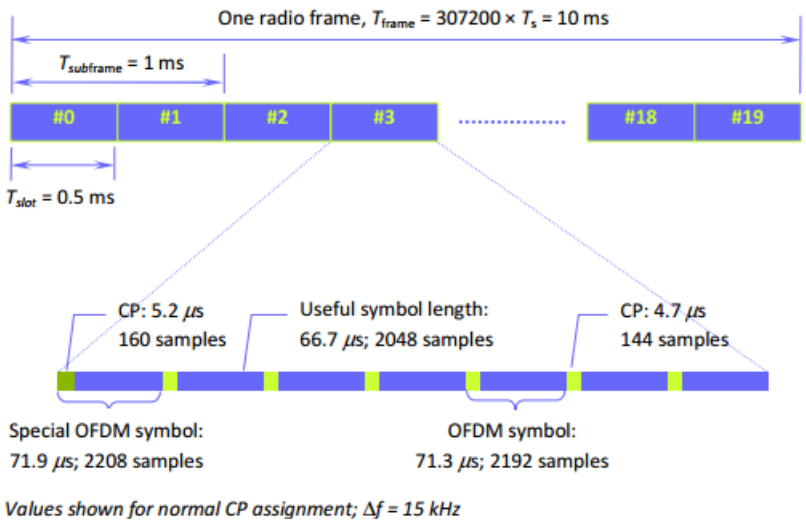


Fig 2. 2 Estructura FDD

El prefix cíclic és un senyal temporal que s'extreu de les últimes mostres de cada subframe i s'introdueixen al principi del cada subframe per evitar les interferències Intersimbòliques (ISI) causades per la propagació multi camí, això fa que el *Delay Spread* sigui inferior que el mateix CP en qualsevol cas. Hi ha dos tipus de CP, el Normal i l'Extens. En el cas del Normal cada Slot està format per 7 símbols OFDM , mentre que en el cas de l'Extens el Slot està compost de 6 símbols OFDM. Això fa que cada CP tingui unes aplicacions associades, en el cas del CP Normal s'utilitza a les cèl·lules urbanes i en aplicacions amb una gran taxa de velocitat, mentre que el CP Extens s'utilitza en casos com *Multi-cell Broadcast*, en cèl·lules molts grans (zones rurals) i en aplicacions de baixa taxa de transmissió.

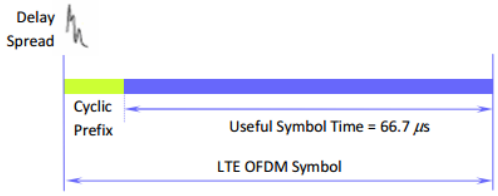


Fig 2. 3 Estructura Prefix cíclic

En el domini freqüencial, el nombre de subportadores associat a cada símbol OFDM varia entre 128 i 2048 depenent de l'Ample de banda associat al senyal (1.4MHz, 3MHz, 5MHz, 10MHz, 15MHz o 20MHz). La separació entre subportadores és de Δf = 15KHz, per tant per saber la freqüència de mostreig associada a cada ample de banda que posseeix LTE, s'hauria d'aplicar la següent relació: $fs=\Delta f * N$, on N és el número de subportadores associades en cada símbol OFDM. El resultat serà múltiple o submúltiple del chip Rate de WCDMA que és de 3.84Mcps.
En LTE el nombre total de subportadores d'un símbol OFDM és potència de 2 per facilitar el processament de la DFT, però no totes les subportadores transporten informació del sistema. Hi ha subportadores que són de padding per tal d'arribar al la potència de 2, aquest s'anomenen portadores de guarda i el seu valor és 0. A més a més, en tots els amples de banda hi ha una portadora central anomenada DC, que està situada al centre.

En LTE ambdós dominis es relacionen de manera que hi ha elements bidimensionals (Temps vs Freq) anomenats Resource Element (RE). Cada element es correspon a una subportadora i a un símbol OFDM. En la transmissió LTE, els RE s'agrupen en blocs anomenats Resource Block (RB) o Physical Resource Block (PRB) que contenen 12 subportadores en una duració d'un Slot (0.5ms).

En el cas del CP Normal cada RB conté 84 RE perquè té 7 símbols OFDM, mentre que en cal del CP Extens cada RB consta de 72 RE.

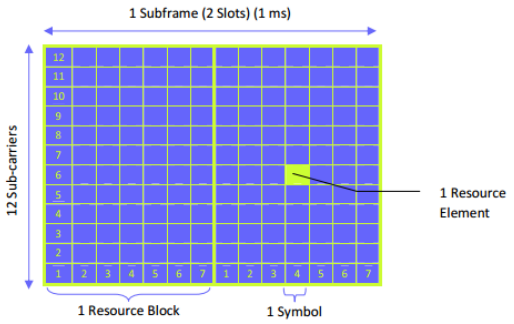


Fig 2. 4 Recursos de LTE

Un resum de les característiques de tots els senyals en LTE es pot veure en la **Taula 2.1**

Ample de banda (MHz)	1.4	3	5	10	15	20
Duració Frame (ms)	10					
Duració sub-frame (ms)	1					
Espai entre sub-portadores (KHz)	15					
FFT	128	256	512	1024	1536	2048
Freqüència de mostreig (Msps)	1.92	3.84	7.68	15.36	23.04	30.72
Núm. d'agrupacions de RB	6	15	25	50	75	100
Núm. Sub-portades ocupades (No inclou DC)	72	180	300	600	900	1200
Núm. Sub-portadores de guarda	56	76	212	424	636	848
Ample de banda ocupada en la TX (MHz)	1.08	2.7	4.5	9	13.5	18
Núm. Símbols OFDM	7 (Normal) / 6 (Extens)					
Durada CP Normal/Extens	5.2 (Primer símbol OFDM) i 4.7 (6 següents símbols OFDM) / 16.67 (tots símbols OFDM)					

Taula 2. 1 Característiques LTE

2.2. Senyals físics

Les senyals físiques de LTE tenen com a objectiu ajudar a rebre satisfactòriament, per part del UE, el senyal LTE transmès per part de l'eNodeB. Hi ha dos tipus de senyals que tenen aquest objectiu i són els senyals de sincronisme i els senyals de referència.

2.2.1. Senyals de sincronisme

Per poder accedir al senyal LTE i poder descodificar les dades enviades per l'eNodeB, tot senyal LTE ha de tenir uns paràmetres de sincronització. Amb la sincronització podrem determinar quan comença el símbol correctament, podrem sincronitzar les portadores en freqüència per evitar el efecte Doppler, i per últim, sincronitzar la freqüència de mostreig del rellotge en transmissió.

La sincronització és molt important per trobar la cèl·lula. És necessària trobar-la tant en un primer moment per alinear el subframe com per mantenir un control de sincronització del senyal, i per detectar las cèl·lules veïnes per si l'usuari ha de fer handover.

Aquest tipus de senyal físic té dos tipus de senyals diferents: Primary Syncronization Signal (PSS) i Secundary Syncronization Signal (SSS). Una vegada l'usuari detecti ambós senyals, s'haurà sincronitzat el senyal tant en temps com en freqüència.

Els dos senyals de sincronització es transmeten cada 5 ms, en els símbols 6 (SSS) i 7 (PSS) en cas del CP Normal, i en els símbols 5 (SSS) i 6 (PSS) en el cas del CP Extens, del Slot 0, del subframe 0 i 5. Les dues ocupen els 6 RB centrals amb independència del ample de banda del senyal, que corresponen a

72 portadores. Ambdues tenen en els seus extrems 5 subportadores que no utilitzen, per tant utilitzen les 62 centrals per enviar informació.

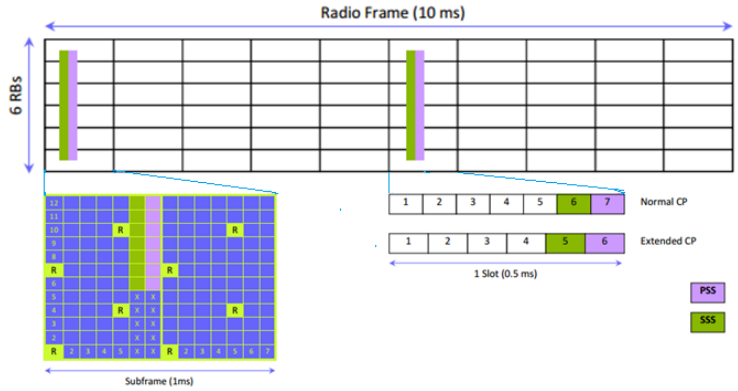


Fig 2. 5 Senyals de sincronisme

El senyal de sincronisme PSS és un senyal codificat per la seqüència Zadoff-Chu (ZF) que proporciona una correlació alta tant en temps com en freqüència. Això es produeix perquè és simètrica en ambdós dominis i utilitza la condició CAZAC[1]. La seqüència ZF pot generar tres tipus de senyals i cadascun indica quin és el número de la cèl·lula (N_id_2) que està transmeten l'eNodeB[3]. En el moment de la recepció, aquest senyal ens permet obtenir la posició de la finestra de la FFT, l'error de freqüència degut a l'efecte Doppler, ens permet sincronitzar-nos a nivell de Slot i extreure el N_id_2 de la cèl·lula en la que està reben amb el residu entre la divisió del Cel_IID i 3.

El senyal de sincronisme SSS és un senyal codificat per la seqüència-m en que s'intercalen dues seqüències binaries pseudo-aleatòries de 31 bits. La primera seq depèn d'un valor m0, mentre que la segona depèn del valor m1. L'ordre de les seq alhora d'intercalar-les és molt important ja que es generen dos senyals diferents. El senyal resultant d'intercalar la seq amb menor valor 'm' en primer lloc és la senyal SSS que es col·locarà en el subframe 0, mentre que el senyal resultant d'intercalar la seq amb major valor 'm' en primer lloc serà la SSS que es col·locarà en el subframe 5. Els valors de les dues seqüències determinen el grup de cèl·lules (N_id_1) en que està l'eNodeB[4]. Gràcies a aquest algoritme, a la recepció, el senyal SSS ens permet sincronitzar-nos de forma complerta a la trama LTE, ens permet extreure el CP associat a la trama, el mode de duplexat (FDD o TDD) i ens permet sincronitzar-nos amb el canal PBCH que posteriorment explicaré.

Els dos senyals de sincronisme combinats formen l'ID de la cèl·lula:

$$Cell\_id = N\_id\_1 * 3 + N\_id\_2 \qquad \textbf{(2.1)}$$

[1] [1] CAZAC és una condició anomendada Constant Amplitud Zero Autocorrelation en que la seva autocorrelació periódica és estrictament nula excepte en el origen.

Tot el procés de codificació i descodificació de les dues senyals es basa en el següent diagrama de blocs:

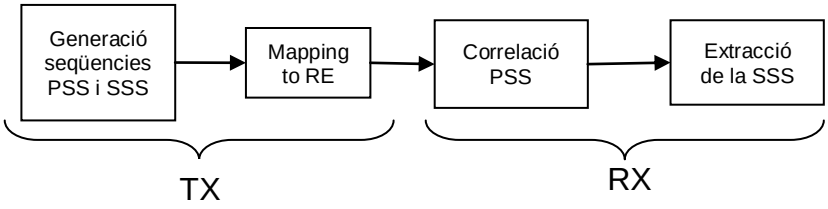


Fig 2. 6 Diagrama de blocs de les senyals de sincronisme

2.2.2. Senyals de referència

Els senyals de referència (RS) coneguts com els senyals pilots, estan repartits en tot el Radio Frame i tenen la finalitat de poder determinar l'estimació del canal i poder determinar el número d'antenes que està transmeten el senyal. La posició d'aquests senyals es fan de manera unitària, és a dir, es col·loquen RE de manera estratègica per cada RB depenent del nombre d'antenes i del Cell_ID que tingui l'estació Base. En la implementació s'explicarà els senyals de referència específics de la cèl·lula.

Les RS es generen a partir d'una seqüència anomenada $r_{l,ns}$[5]. La posició dels senyals de referència ve marcada pel nombre d'antenes que utilitzarà l'eNodeB en la transmissió del senyal. Tot i així, cada antena ocupa 4 RE en cada RB, i la separació entre aquests és de 90 KHz. En la **Fig 2.6** es marquen les posicions dels RS quan tenim una, dues antenes en transmissió.

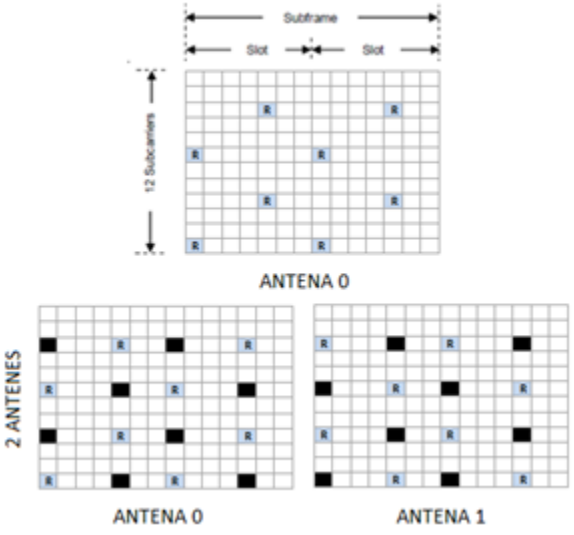


Fig 2. 7 Posicionament senyals RS

Tot el procés de codificació i descodificació dels senyals de referència es basa en el següent diagrama de blocs:

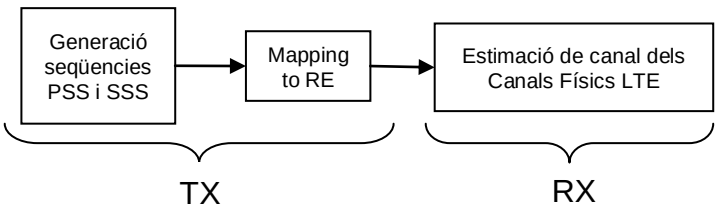


Fig 2. 8 Diagrama de bloc de les RS

2.3. Canals de la capa física del Downlink

En LTE la capa física deu ser suficientment flexible per acomodar de forma dinàmica les necessitats de transmissió dels diferents usuaris, els quals, també se'ls hi ha de dedicar els recursos necessaris en cada transmissió LTE. En cada canal se'ls ha de proporcionar els suficients recursos i protecció per poder ser transportat als UEs finals. A continuació detallarem els canals físics corresponents al Downlink.

2.3.1. Physical Broadcast Channel (PBCH)

El PBCH transporta tota la informació necessària per tal de que el UE pugui adaptar-se al senyal transmès per l'eNodeB. Aquesta informació està codificada en el Master Information Block (MIB) en un total de 24 bits:

- 3 bits indiquen l'Ample de Banda, i per tant, el número de RB del senyal transmès.
- 3 bits indiquen la informació referent a PHICH:
  - 2 bits indiquen la longitud, si és normal o extensa.
  - 1 bit indica el tipus de recursos (1/6, ½, 1 o 2)
- 8 bits indiquen el número de frame del sistema.
- 10 bits estan reservats.

Aquest canal físic està situat en el domini temporal en els símbols OFDM 0, 1, 2 i 3 del slot 1 del subframe 0, i en el domini freqüencial ocupa les 72 portadores centrals (6 RB), sense utilitzar les subportadores destinades als senyals de referència o RS. Cada transmissió del PBCH es fa cada 10ms i s'envia el mateix MIB durant 40ms o 4 Radio Frames.

2.3.1.1. Procés de codificació

El procés de codificació del PBCH es basa en el diagrama de blocs de la **Fig 2. 8**

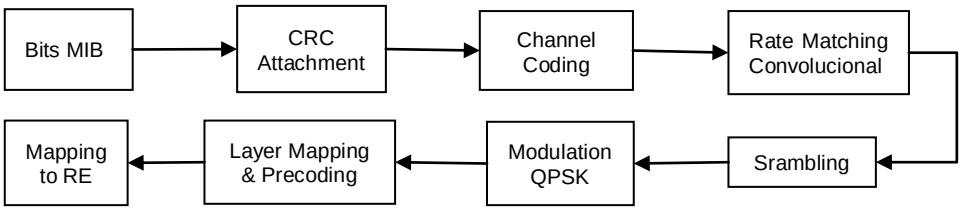


Fig 2. 9 Procés de codificació del PBCH

El CRC concatena 16 bits de paritat al bits del MIB i el passa al codificador Convolucional Tail biting [6]. Passa per tots els blocs del diagrama fins arribar a introduir els símbols QPSK dins de subframe que li correspon.

2.3.1.2. Procés de descodificació

El procés de descodificació es basa en extreure els bits del MIB per tal de poder definir les característiques del senyal LTE rebut i sincronitzat prèviament amb els senyals de sincronisme PSS i SSS.

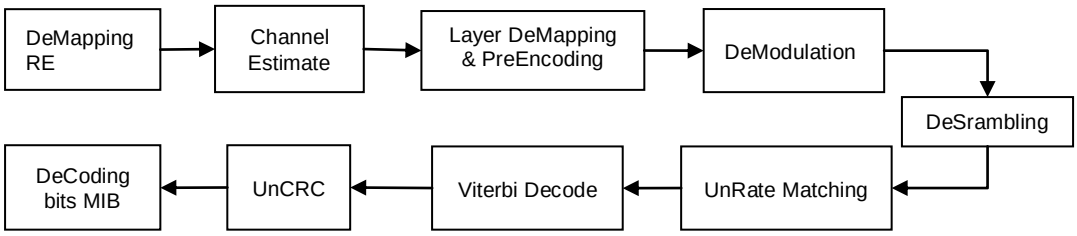


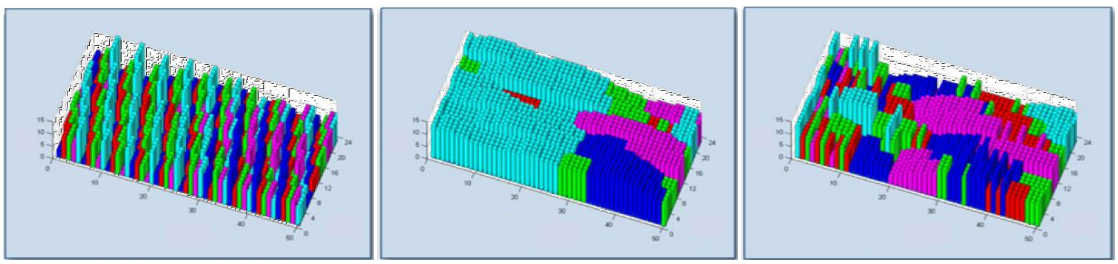
Fig 2. 10 Procés de descodificació PBCH

La gran majoria de passes que s'han de seguir en aquest procés són la inversa del procés de codificació però cal especificar que l'estimació del canal es fa gràcies als senyals de referència.

2.3.2. Physical Downlink Shared Channel (PDSCH)

El PDSCH és el canal principal on estan codificades les dades que se li assignen als usuaris finals. Hi ha diferents tipus d'assignacions dels recursos als usuaris que poden dependre de la QoS (Quality of Service) requerida pel servei:

- **Round Robin**: és simple i no és oportunista, és a dir, cada usuari té els mateixos recursos.
- **Millor CQI**: els recursos són per l'usuari amb millor relació SNIR, aquest tipus d'assignació o Scheduling maximitza el rendiment de les cèl·lules però en canvi dóna pitjor servei a l'usuari i provoca insatisfacció.
- **Proportional Fair:** Els usuaris se'ls hi dóna una prioritat tenint en compte el rendiment instantani i el rendiment mitjà del passat en un interval de temps.[7]



(A) (B) (C)

Fig 2. 11 Sheduling. (A) Round Robin, (B) Millor CQI i (C) Proportional Fair

Aquest canal físic porta les dades en els Blocs de Transport (BT, Block Transport) [8] que es corresponen a una MAC PDU. Aquests passen de la capa MAC a la capa PHY una vegada cada TTI (Interval de temps de transmissió) que es correspon a 1ms per complir els requeriments de baixa latència.

2.3.2.1. Procés de codificació

En cada Transport block s'envien bits d'informació al UE, però aquest pot ser diferent per cada UE depenent de diversos paràmetres com el QoS o el Scheduling. Aquests bits passen per diferents etapes abans de ser introduïts en el subframe i posteriorment ser enviats al UE o als UEs finals. El procés és el següent:

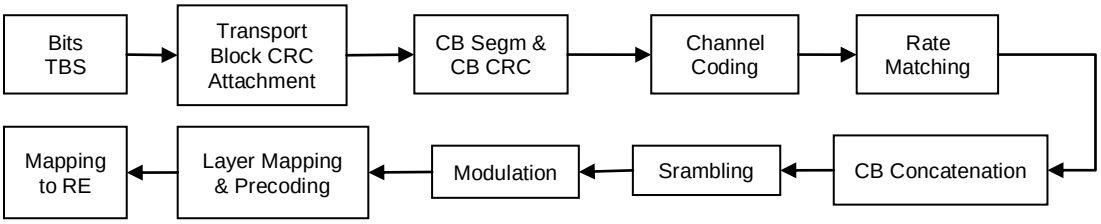


Fig 2. 12 Procés codificació PDSCH

Cada TB té assignat un número de bits a enviar depenen de la modulació que en que es vulgui enviar la informació, i aquesta modulació pot ser de tres tipus: QPSK, 16 QAM i 64 QAM.

Aquests bits passen al bloc 'Transport Block CRC Attachment' on es concatenen 24 bits de paritat al final de la seqüència de bits del TB per evitar errors de detecció. A continuació, s'entra al bloc 'Code Block Segmentation & Code Block CRC' el qual s'utilitza per evitar que els blocs de bits siguin molt grans i sigui molt costós de codificar en termes de computació. Per aquest motiu si després del TB CRC es supera una quantitat de 6144 bits, llavors s'agafen blocs de 6144 bits i es torna a concatenar un CRC de 24 bits[9]. A continuació es codifiquen els blocs de bits individualment en un Turbo Coder amb un Tail de 12 bits[10]. A cada bloc se l'hi aplica un Rate Matching turbo amb una code rate de 1/3 i seguidament es concatenen tots els blocs en cas d'haver-se segmentat anteriorment[11].

Una vegada tots els bits d'informació han estat tractats i concatenats en el cas d'una segmentació prèvia, passen pel procés de 'Scrambling', de 'Modulació', 'Layer Mapping & Precoding' i finalment s'introdueixen en els RBs corresponents del subframe, prèviament assignats.

2.3.2.2. Procés de descodificació

El procés de descodificació del canal PDSCH és el procés invers al codificador però introduint l'estimador de canal abans d'extreure els símbols del subframe. Aquest mòdul ajuda a recuperar els RE després de passar pel canal de comunicació. Els REs els extraiem després de que el senyal LTE s'ha sincronitzat, ha extret el MIB, ha configurat el senyal per tal de rebre l'Ample de Banda que ens proporciona el MIB, ha descodificat els canals de control i s'ha informat a l'usuari de quins RB ha de descodificar.

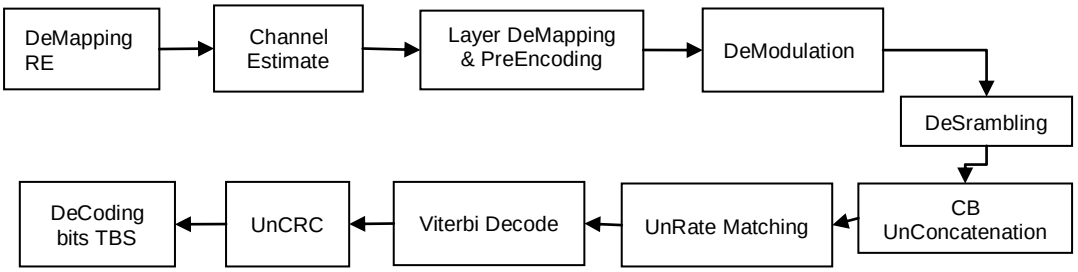


Fig 2. 13 Procés de descodificació del PDSCH

2.4. Canals de control de la capa física del Down-link

Els canals de control de la capa física són els encarregats d'assignar els recursos disponibles a cada usuari i de determinar quins recursos de la trama són els encarregats de transportar la informació de control. El canals de la capa física que s'encarreguen d'aquestes taques són el PCFICH i el PDCCH, a continuació hi ha una explicació més extensa.

2.4.1. Physical Control Format Indicator Channel (PCFICH)

El canal PCFICH transporta l'Indicador del Format de Control (CFI), aquesta informació fa referència al número de símbols OFDM que utilitzen els canals de control PDCCH i PHICH en tots els subframes de les trames. Depenen del Resource Block escollit en l'eNodeB, el CFI s'aplica en diferents posicions del Slot 0. En el cas de tenir un RB≤10 els símbols que ocupen els canals de control seran a partir del símbol OFDM 1, mentre que si RB>10 els símbols de control començaran a partir del símbol OFDM 0.

El canal PCFICH utilitza 16RE agrupats en quatre quartets de 4REs o REG (Resource Element Group), tots ells en el símbol 0 del Slot 0 de cada sub-trama. La ubicació del quartets estan distribuïts de manera uniforme a través de tot l'ample de banda, independentment de quin sigui l'Ample de Banda del senyal per tal d'aconseguir diversitat en freqüència. Tot i així, la posició exacte del canal depèn del BW i del ID de la cèl·lula.

Una vegada sabem els CFI, el canal passa per un procés de codificació fins introduir els símbols de la modulació al subframe.

2.4.1.1. Procés de codificació

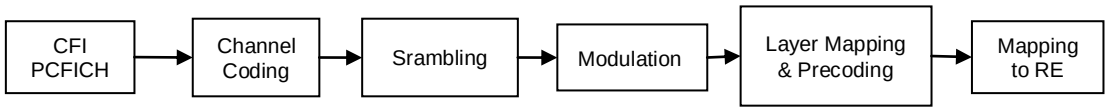


Fig 2. 14 Procés codificació PCFICH

El procés de codificació consisteix a codificar una sèrie de 32 bits depenen del valor de CFI que transporti el canal PCFICH. Segons el tipus de CFI els bits que es generen en la codificació són diferents als altres, a continuació es mostra una taula que indiquen els bits en funció del CFI.[12]

CFI	CFI CodeWord (bits)
1	<0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1>
2	<1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0>
3	<1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1>
4 (Reserved)	<0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0>

Taula 2. 2 CFI Bits

Una vegada sabem els bits codificats passem al mateix procés que els canals físics del Downlink, on actua el Scrambling, el 'Modulation' que en aquest cas es modula en QPSK, 'Layer Mapping & Precoding' i per últim, el bloc 'Mapping to RE' es fa el *mapping* dels símbols QPSK com hem explicat anteriorment amb els quartets de 4RE.

2.4.1.2. Procés de descodificació

El procés de descodificació és el procés invers al codificador però introduint el estimador de canal, abans d'extreure els símbols del subframe. Aquest módul ajuda a recuperar els RE després de passar pel canal de comunicació. El RE els extraiem després de que el senyal LTE s'ha sincronitzat, ha extret el MIB i ha configurat el senyal per tal de rebre l'Ample de Banda que ens proporciona el MIB.

2.4.2. Physical Downlink Contol Channel (PDCCH)

El canal PDCCH és un canal de control que té com a finalitat transportar el DCI (Downlink Control Information), donar informació sobre quins són els recursos associats a cada usuari, tant en Downlink com en Uplink, i altres dades de control.

El missatge DCI transmet informació de control rellevant del sistema LTE, aquesta informació conté tot el necessari per tal que els UEs siguin capaços d'identificar els recursos necessaris per rebre el PDSCH en la subtrama i descodificar-lo. El DCI té una sèrie de formats que depenen de la configuració de les antenes, de la configuració del sistema LTE i del mode de transmissió del PDSCH.

Format DCI	Aplicació
0	Assignacions pel PUSCH (Physical Uplink Shared Channel)
1	Assignacions pel PDSCH quan es transmet en mode simple
1A	Assignacions pel PDSCH quan es transmet en mode compacte
1B	Assignacions pel PDSCH en mode MIMO
1C	Assignacions pel PDSCH en format molt compacte
1D	Assignacions pel PDSCH en mode MU-MIMO
2	Assignacions pel PDSCH en mode MIMO amb enllaç tancat
2A	Assignacions pel PDSCH en mode MIMO amb enllaç obert
3	Comandaments de control de potència pel usuaris en el PUCCH i PUSCH en 2 bits.
3A	Comandaments de control de potència pel usuaris en el PUCCH i PUSCH en 1 bits.

Taula 2. 3 Format DCI [14]

Una altra informació important que transporta el canal és el RNTI (Radio Network Temporaly Identifier). Aquest és un número que identifica canals de radio específics i diferents usuaris. A més a més s'utilitza per codificar el CRC dels missatges del PDSCH, això implica que sense aquest identificador l'UE no pot descodificar els missatges del canal Radio, encara que el destinatari sigui l'UE. L'eNodeB enviarà diferents identificadors depenen del moment en que estigui l'UE. En la següent taula es pot veure alguns tipus d'informació esdevenen dels identificadors RNTI.

Format RNTI	Abreviatura RNTI	Aplicació
Paging RNTI	P-RNTI	S'utilitza per la transmissió de missatges de Paging.
System Information RNTI	SI-RNTI	S'utilitza per la transmissió de missatges SIB (System Information Block)
Random Acces RNTI	RA-RNTI	S'utilitza per la transmissió de les respostes del PRACH
Cell RNTI	C-RNTI	S'utilitza en les transmissions de dades a un UE
Temporaly RNTI	T-RNTI	S'utilitza en les transmissions durant el RACH

Taula 2. 4 Format RNTI [14]

Aquest canal es transmet en tots els sub-frames de la trama. En cada sub-frame es pot transmetre un o més d'un PDCCH, això vol dir que la informació dintre del canal pot estar destinada a un sol usuari o a varis alhora. La seva posició dintre del subframe és similar a la del canal PCFICH i utilitza l'agrupació REG, però en aquest cas cada PDCCH utilitza un número de CCE (Control Channel Elements) igual a 1, 2, 4 o 8, on cada CCE és la unitat d'assignació dels recursos de control y conté 9 REG (72 bits). El número de CCE que s'utilitza a cada sub-frame ve determinat per la SNR que necessita.

2.4.2.1. Procés de codificació

Per tal de codificar el canal PDCCH, els bits generats segons el format del DCI passen per un procés de codificació per tal de ser introduïts en el subframe i és el següent:

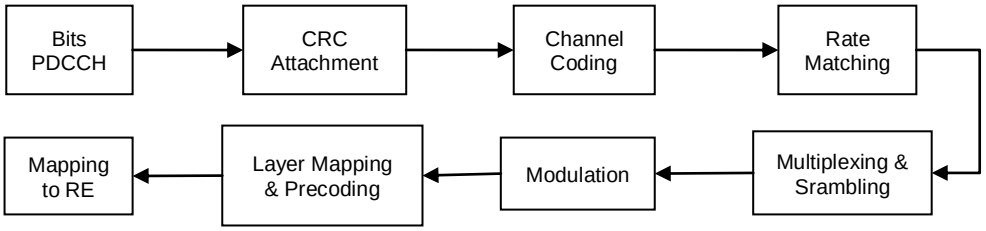


Fig 2. 15 Procés de codificació del PDCCH

Els bits d'informació que transporta el canal PDCCH dependrà del format de DCI que s'assigni en la transmissió.

PDDCH, DCI Format	Bits del canal PDCCH
0	72
1	144
2	288
3	576

Taula 2. 5 Bit en funció del format DCI

Una vegada se sap el número i els bits que conté el format del DCI, aquests passen al CRC on es concatenen 16 bits de paritat al final de la seqüència, a més a més al bits resultants se'ls hi aplica un Scrambling de bits corresponents al identificador RNTI. Seguidament es codifiquen els bits amb el codificador '**Tail Biting Convolutionally code'** pel Rate Matching convolucional amb un code rate de 1/3 cadascun.[13] Seguidament tenim els mateixos blocs que en els anteriors canals amb la diferència que la modulació dels bits es fa amb una

QPSK, i que el 'Mapping to RE' depenen del format del DCI. Depenen del format es poden agrupar els símbols en diferents grups.

PDDCH, DCI Format	# CCE	# REG	Símbols en 1 REG	Bits en 1 Símbol	Bits del canal PDCCH
0	1	9	4	2	72
1	2	18	4	2	144
2	4	36	4	2	288
3	8	72	4	2	576

Taula 2. 6 Informació del DCI

Les posicions del CCE es reparteixen pels RE associats al canal PDCCH. Els símbols OFDM que ocupa el canal depenen de la informació (CFI) que transporta el canal PCFICH i com s'ha explicat en **l'apartat 2.4.1** en el cas de tenir un RB≤10 els símbols que ocupen els canals de control seran a partir del símbol 1, mentre que si RB>10 els símbols de control començaran a partir del símbol OFDM 0.

2.4.2.2. Procés de descodificació

El procés de descodificació del canal PDCCH és el procés invers al codificador però introduint el estimador de canal, abans d'extreure les mostres del subframe. Aquest mòdul ajuda a recuperar els RE després de passar pel canal de comunicació. El RE els extraiem després de que el senyal LTE s'ha sincronitzat, ha extret el MIB, ha configurat la senyal per tal de rebre l'Ample de Banda que ens proporciona el MIB i ha extret el valor del DCI en el canal de control PCFICH.

CAPÍTOL 3. DESENVOLUPAMENT DE LTE

En aquesta part del projecte, s'explicarà les passes a seguir per tal de poder implementar l'eNodeB. L'objectiu és implementar una estació base LTE a l'enllaç DownLink introduint tots els senyals físics i els canals físics que s'han vist al capítol anterior. A més a més, s'implantarà un UE per tal de rebre les dades enviades per l'eNodeB i seguir tot el procés de descodificació que segueixen els UEs en l'enllaç descendent.

Per tal de poder aconseguir l'objectiu, s'explicarà tot el procés d'implementació per separat de l'eNodeB i d'un UE i, seguidament, s'explicarà com s'implementen ambdós sota l'entorn ALOE per tal de poder aprofitar els avantatges de multiprocessament que ens proporciona la tecnologia SDR.

També s'utilitzarà una llibreria Open Source anomenada SRSLTE que proporcionarà codi LTE. Aquesta llibreria, al ser Open Source, s'anirà tractant al llarg de la implementació, i per tant, es faran les modificacions i correccions adients.

3.1. Implementació de l'eNodeB

La implementació de l'eNode té com a objectiu crear el Frame Radio par tal de ser enviat al usuari. Per aconseguir-ho s'ha de tenir present quines són les passes que s'han de complir.

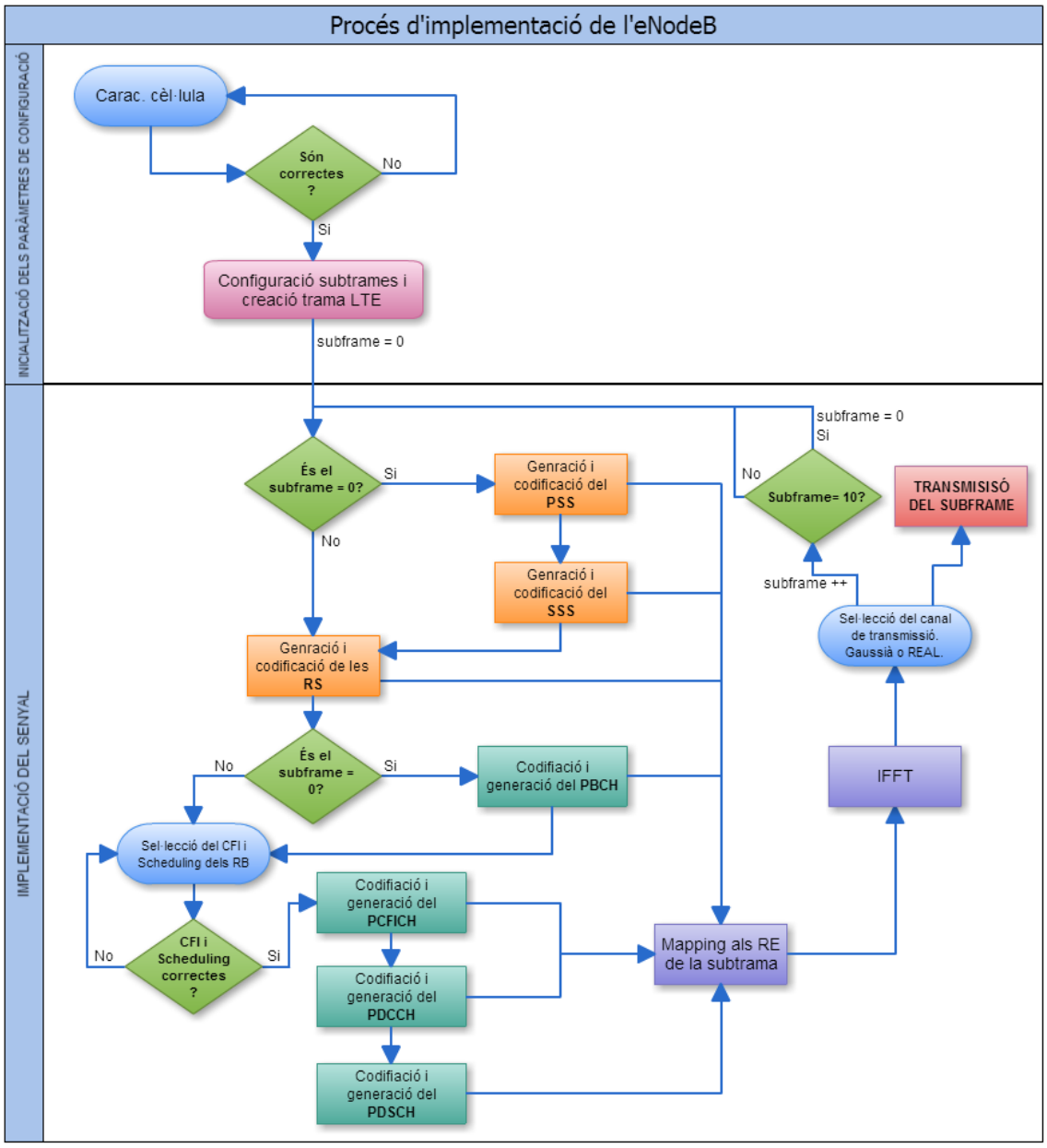


Fig 3. 1 Diagrama de flux de la implantació de l'eNodeB

Primerament, el que s'ha de fer és determinar quines són les característiques de la cèl·lula que transmet l'eNodeB. Aquestes característiques vénen determinades pel número de RB, el nombre de ports, l'Identificador i el prefix cíclic del senyal.

El número de RB marca l'Ampla de banda que utilitza el senyal en el domini freqüencial i ens determina la freqüència de mostreig que s'assigna a la USRP. El nombre d'antenes del senyal determina els senyals pilots o RS de la trama i, a més a més, marca el tipus de transmissió que es fa del senyal (SISO, MIMO, SIMO...). El prefix cíclic determina el número de símbols OFDM que posseeix cada Slot. I per últim, l'identificador de la cèl·lula indica quin dels tres tipus de

PSS s'utilitzarà per poder oferir una bona sincronització, i ens indica els valor m0 i m1 de la seqüència-m per tal de generar la SSS.

Seguidament s'ha configurar les subtrames on s'introduirà tota la informació en cadascun del REs i es crea la trama LTE. Per saber el nombre total de REs de cada subframe s'ha d'aplicar la següent fórmula:

$$sf\_n\_RE = 2 * CP\_símbols\_Slot * nof\_RB * RE\_n\_RB \quad \textbf{(3.1)}$$

On '2* CP_símbols_Slot' és el número de símbols en subframe en un TTI (1ms), mentre que 'nof_RB * RE_n_RB' és el número de RE en un símbol OFDM.

La introducció dels senyals de sincronisme i dels canals físics de LTE s'ha de fer **contínuament en cada subframe,** de forma esglaonada i seguir sempre el diagrama de flux de la **Fig 3.1**.

3.1.1. Senyals de sincronisme

Una vegada està configurada la trama, s'ha de mirar en quin subframe està el sistema. En cas d'estar en el subframe 0 i 5, es generen i s'introdueixen els senyals dins el suframe. Per saber quins senyals de sincronisme esdevenen del Cell_ID, simplement s'ha d'extreure el residu entre el Cell_ID i el nombre total de PSS (3). Aquest valor proporciona el N_id_2, que permet determinar quina de les següents PSS s'introdueixen al subframe.

Per tal de poder codificar la PSS com ens indica **l'apartat 2.2.1** , s'ha d'extreure el valor de N_id_1 de la fórmula (**2.1)**. Amb aquest valor es pot saber els valors 'm0' i m'1' de la seqüència-m amb la taula de l'estàndard 3GPP[4].

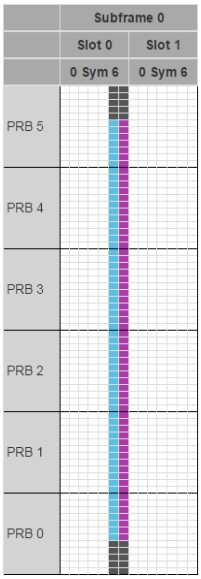


Fig 3. 2
Mapping PSS i SSS

3.1.2. Senyals de referència

Seguidament es codifiquen i s'introdueixen en tots els subframes de la trama, els senyals de referència. El número d'aquests senyals depèn del número d'antenes per les quals es transmet el senyal. El procés de codificació que segueixen aquests senyals es correspon al procés descrit en **l'apartat 2.2.2**. Un aspecte important és la relació entre el número d'antenes i el número de REs que ocupa el senyal.

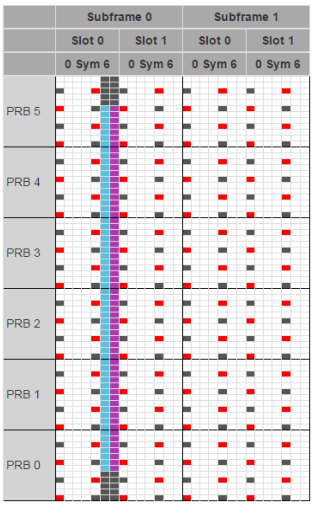


Fig 3. 3 *Mapping* dels RSs

3.1.3. Physical Broadcast Channel (PBCH)

Un cop s'han introduït els senyals de referència en el subframe de trama física de LTE, s'ha de tornar a mirar quin subframe està codificant. En el cas de ser el subframe 0, és el moment de codificar aquest canal de la capa física tal com s'ha explicat en **l'apartat 2.3.1.**

Els 24 bits d'informació que transporta el canal de la capa física s'extreuen de les característiques de la cèl·lula introduïdes a l'inici de la implementació del eNodeB:

- 3 bits indiquen l'Ample de Banda, i per tant, el número de RB del senyal transmès.
- 3 bits indiquen la informació referent a PHICH:
  - 2 bits indiquen la longitud, si és normal o extensa.
  - 1 bit indica el tipus de recursos (1/6, ½, 1 o 2)
- 8 bits indiquen el número de frame del sistema.
- 10 bits estan reservats.

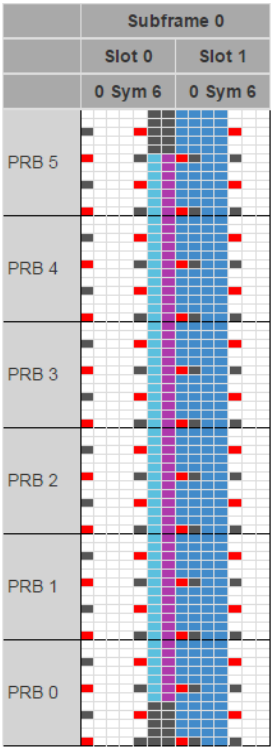


Fig 3. 4 *Mapping* del PBCH

3.1.4. Physical Control Format Indicator Channel (PCFICH) i Physical Downlink Control Channel (PDCCH)

Els següents canals de la capa física que s'han d'introduir en el subframe són els de control. Primerament, s'escull un CFI entre 1 i 3 per tal de determinar

quants símbols OFDM ocuparà el canal de control PDCCH, i després es codifica el canal PCFICH seguint les passes descrites **l'apartat 2.4.1**.

Seguidament s'ha de determinar el nombre d'usuaris en recepció i els recursos que se'ls hi assigna, és a dir, fer Scheduling. En cas nostre s'implementa només un únic usuari perquè es tracta d'avaluar el funcionament complet del sistema assignant més o menys recursos. Per aquest motiu se l'hi assigna la totalitat dels RBs al receptor.

Com es veu a la **Taula 2.2** el format del DCI depèn del mode del sistema, en el cas que s'està implementant s'assignen els recursos del PDSCH en mode simple i per tant, el format DCI és l'1. Aquest format fa que el canal hagi que codificar 144 bits d'informació en 2 CCE consecutius dins de cada subframe de la trama. Cal tenir en compte que si es vol, es pot variar la quantitat RB assignat al UE.

Un altre terme important en la implantació del PDCCH en la trama LTE, és el RNTI. Aquest indica quin tipus de dades enviarà el senyal segons les aplicacions que es vulgui utilitzar les dades. Pot haver-hi identificadors broadcast o específics per usuari.

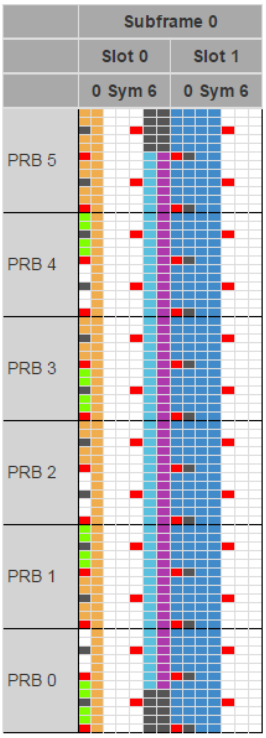


Fig 3. 5 *Mapping* PCFICH i PDCCH

3.1.5. Physical Downlink Shared Channel (PDSCH)

Per últim, s'ha d'implementar el canal de la capa física responsable de transportar les dades cap a l'UE, el PDSCH. Les dades que transporta cada RB estan determinades pel TBS i per la Modulació que s'utilitzi en aquell moment. Aquests dos factors fan que es determini el MCS, que relaciona la modulació (QPSK, 16QAM i 64QAM) amb el nombre de bits del TBS. Els símbols complexes extrets de la codificació del canal s'agrupen en els RBs que prèviament ha informat el PDCCH per tal de ser assignats als seus UE.

En aquest moment el sistema ha introduït tots els canals físics i els senyals en el subframe LTE per qualsevol BW. Un exemple visual de com queden agrupats tots els senyals i els canals de la capa física es pot veure l'**Annex 1**

Com ens indica el diagrama de flux de la **Fig. 3.1**, un cop es tenen les mostres en el subframe el següent pas és aplicar la IFFT.

3.1.6. IFFT

Per tal d'enviar la trama al UE o als UEs, el que s'ha de fer és convertir les subportadores freqüencials al domini temporal. Per obtenir les mostres en domini temporal s'ha d'aplicar una IDFT a les mostres d'un símbol OFDM, però com que el procés de la IDFT és molt costós computacionalment, s'ha d'utilitzar

el mecanisme IFFT. En la IFFT s'ha d'augmentar el nombre de mostres a 2^n amb símbols complexes igual a 0, les denominades 'Portadores de guarda'.

Les 2^n subportadores que hi ha en símbol OFDM es posicionen de la següent forma abans d'aplicar IFFT:

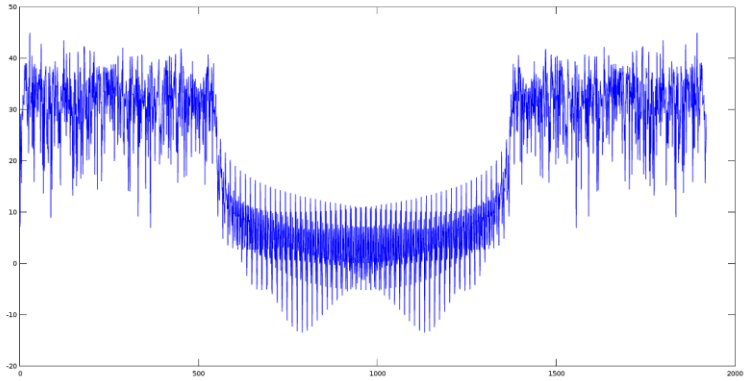


Fig 3. 6 Espectre del senyal des de 0 fins a fs/2

On l'espectre del senyal està situat de 0 a fs/2 ja que la IFFT en software no accepta índexs negatius quan s'està implementant en arrays. Per aquest motiu agafa les subportadores imatges en fs/2.

Una vegada s'ha fet la IFFT al senyal anterior, s'afegeix el CP que s'ha escollit de la cèl·lula de l'eNodeB. Tot aquest procés es repeteix amb els 13 símbols OFDM restants fins tenir un senyal temporal on el número de mostres ve determinat per la següent equació:

$$sf\_n\_samples = 2 * Slot\_length(samples\_símbol) \quad \textbf{(3.2)}$$

On el 'Slot_length' és una funció que extreu el número del Slot en funció del número de mostres d'un símbol OFDM.

3.1.7. Tipus de transmissió del senyal

Per tal d'enviar el senyal LTE s'ha implementat dos tipus de sortides. A la primera, s'agafen les dades temporals resultants i s'aplica un canal AWGN simulat, i seguidament es guarden les dades en un fitxer. En aquest cas es pot variar el senyal per tenir una determinada SNR. A la segona opció s'introdueix una USRP per tal d'enviar les dades a través del canal real.

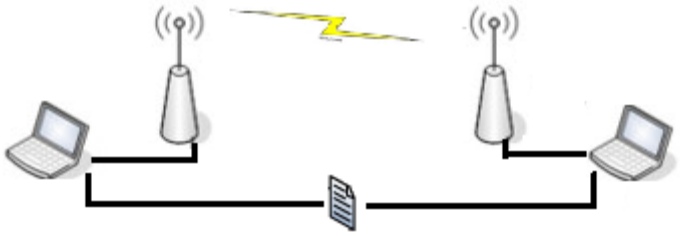


Fig 3. 7 Canals de transmissió de l'eNodeB

En cas d'introduir una USRP a la transmissió de les dades, s'ha de comprovar que la fs del sistema LTE és la mateixa que la que pot suportar la USRP. En cas de no coincidir, la USRP escull la f_s més propera de les seves configuracions. Un exemple és el següent: el senyal LTE que volem transmetre ocupa un BW de 10MHz. Aquest ample de banda té associat una f_s de 15.36Msps però la USRP no pot associar la seva f_s interna a del senyal LTE. Per tant, comprova el seu sistema i escull una f_s de 16.66Msps.

Aquest fet provoca que el senyal enviï més mostres de les que es vol i que aquestes no siguin les correctes. Per aquest motiu, una solució parcial per mantenir el nombre de mostres a la transmissió és augmentar la separació entre les subportadores fins a 16276Hz.

En l'Annex 3 està detallat el codi en llenguatge C que implementa tot el que s'ha explicat en aquest apartat.

Per tal d'aconseguir els resultats satisfactòriament, s'ha hagut de modificar certs aspectes de la llibreria srsLTE que no funcionaven correctament. Aquests errors provenien, exclusivament, del *mappping* dels símbols complexes a les subportadores dels símbols OFDM. Hi ha tres casos que el *mappping* no era correcte: en el cas dels senyals de sincronisme, no es col·locaven els 0 en els extrems del senyal, en el canal PBCH es col·locaven els símbols complexes en els primers 4 símbols del Slot 0, i no en la seva posició correcta com són els 4 primers símbols del Slot1 del subframe 0. I per últim, el canal PDSCH no codificava bits d'informació en els subframes 0 i 5 ja que en aquests s'havia d'aplicar un algoritme diferent per culpa dels senyals PSS i SSS i el canal físic PBCH.

3.1.8. L'eNodeB en ALOE

En aquest apartat l'objectiu és fer funcionar tot el sistema anterior de transmissió LTE dividit en mòduls i aprofitar el multiprocessament que ens proporciona ALOE.

En ALOE cada mòdul està dividit en un procés d'inicialització de les variables i un procés d'execució i processament del senyal que li arriba d'altres mòduls. Per tant, s'ha de donar un cop d'ull a la implementació anterior i separar la part d'inicialització de les variables i la part de processament del senyal.

Normalment la part d'inicialització de les variables és el moment quan s'inicialitzen les variables de tots els objectes dels senyals i canals de la capa física de LTE. La part de processat del senyal es dóna en els processos de codificació, la posterior introducció dels símbols modulats als RE del subframe i el pas per la IFFT + CP del senyal al domini temporal.

Una vegada s'ha determinat les dues etapes dins de la implementació de l'eNodeB en C, el següent pas és determinar els temps d'execució a la part del processat del senyal. Aquests temps ens determinen si el senyal compleix els requisits del TTI = 1ms o si s'ha de distribuir la implementació en mòduls d'ALOE per tal de prevenir aquest error.

BW	TEMPS DELS SUBFRAMES (mitjana)
1.4 MHz	0.18 ms
3 MHz	0.35 ms
5 MHz	0.5 ms
10 MHz	1.8 ms
15 MHz	3.3 ms
20 MHz	6.9 ms

Taula 3. 1 Temps processat de l'eNodeB

En la **taula 3.1** es pot veure els temps de processat amb els diferents Amples de Banda executats amb un processador Intel Core 2 Duo amb una freqüència de 2.2GHz. En els casos en que el temps de processat de cada subframe estiguin per sota del llindar de 1ms, significa que compleixen els temps de TTI que marca l'estàndard LTE. En els altres casos es podria fer proves amb ordinadors amb un millors processadors, amb aquestes proves es podria determinar si no cumpleix l'objectiu pel fet de executar el codi en processadors poc eficients o si és per la implementació.

Per tal d'intentar solucionar els temps de processat s'ha de crear tres mòduls software en ALOE, que basa el seu funcionament en un procés de pipelinnig. El primer serà l'encarregat de codificar els senyals i els canals físics LTE del DownLink i posteriorment introduir-los en el subframe corresponent. El segon mòdul serà l'encarregat de realitzar la IFFT del subframe i passar el senyal del domini freqüencial al domini temporal i, a més a més, introduir el CP associat. I per últim, el tercer mòdul és l'encarregat d'agafar el senyal temporal i l'enviar-lo a través de la USRP.

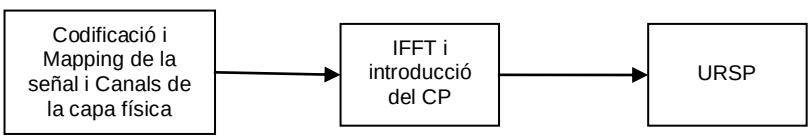


Fig 3. 8 Procés d'execució de l'eNodeB en ALOE

El mòdul de la USRP accedeix al hardware i utilitza la FPGA, els DACs, la part de RF, etc.., de la pròpia USRP. Un dels elements rellevants és el DUC (Digital Up Converter) on es converteix el senyal de banda base a la freqüència portadora que s'ha escollit.

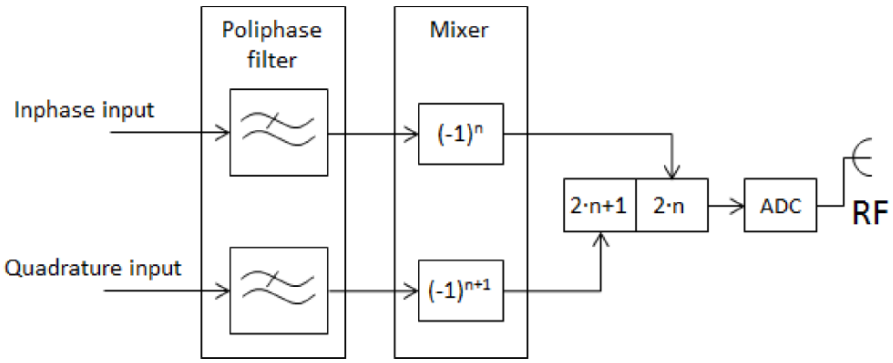


Fig 3. 9 Esquema Digital Up Converter FPGA USRP

3.2. Implementació de l'UE

La implementació de l'UE té com a objectiu descodificar les dades d'informació dels subframes que li arriben de l'eNodeB. Per tal d'assolir l'objectiu s'ha de tenir present quines són les passes que s'han de complir.

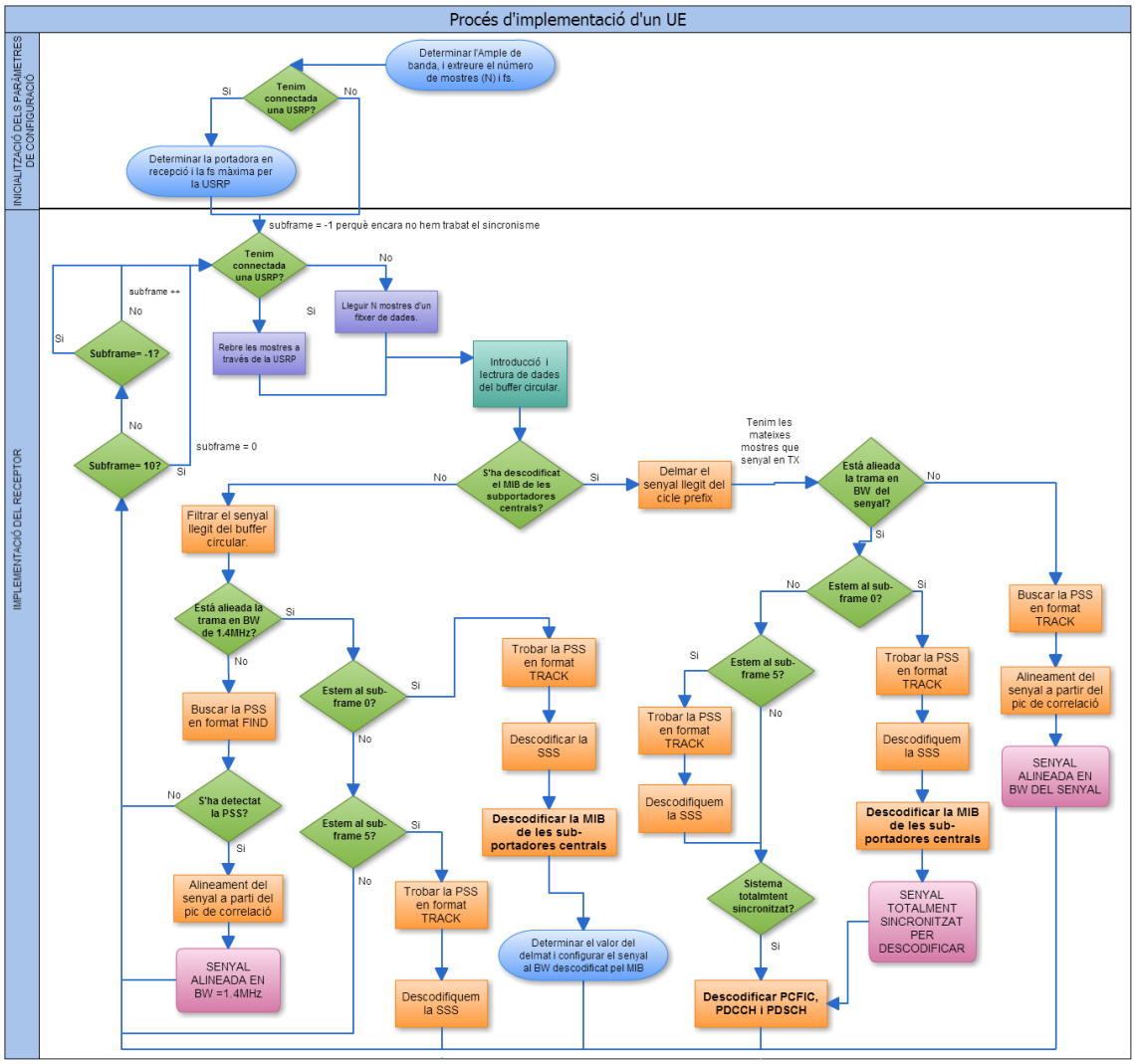


Fig 3. 10 Diagrama de flux de la implantació de l'UE

En la implementació de l'UE s'explicarà el procés que s'ha seguit per descodificar el senyal provinent de l'eNodeB passa per passa.

En primer lloc, en el moment que l'usuari vol rebre un senyal LTE només coneix la freqüència portadora en que s'està enviant. Per aquest motiu l'UE ha d'adaptar el senyal com si estigués rebent sempre l'Ampla de banda més gran.

En el sistema receptor s'ha considerat que BW més gran és 10MHz i no 20MHz com marca l'estàndard, per les limitacions de la USRP. Aquestes limitacions obliguen a baixar la resolució de quantificació de 16 bits a 8 bits perquè no suporta freqüències de mostreig tan elevades i es perd qualitat.

Per començar la implementació del receptor s'ha de decidir entre dos modes de recepció com tenim en la **Fig 3.7.** El primer mode de recepció està implementat per llegir el senyal des d'un fitxer de dades en el nostre sistema. Mentre que el segon mode de recepció està basat en la USRP.

En el segon cas cal determinar la configuració de la USRP a la freqüència de mostreig més alta de LTE. Com s'ha dit abans, en el nostre cas l'Ample de banda màxim és de 10MHz. Aquest BW li correspon una f_s de 15.36Msps per rebre 15360 mostres cada 1 ms.

Igual que a la implementació de l'eNodeB, la USRP no pot assignar una f_s de 15.36Msps, i automàticament utilitza una f_s de 16.66MHz. Per tal de poder adaptar el sistema i mantenir el nombre de mostres corresponents a 10MHz (15360 mostres), el que es fa és augmentar la separació entre les portadores a una diferència de 16276 Hz. Encara que aquesta configuració no segueix l'estàndard de LTE resol parcialment el problema de mostreig, ja que si es manté la configuració es podria arribar a tenir més mostres de les desitjades.

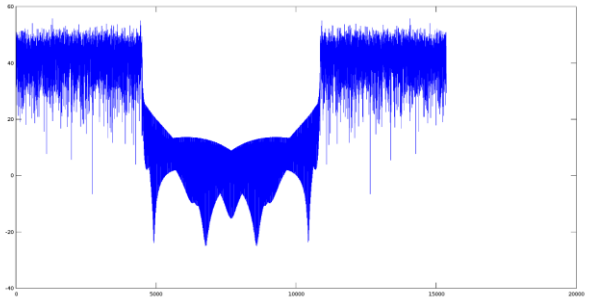


Fig 3. 11 Senyal rebuda amb un BW de 10MHz

Una vegada s'ha mostrejat el senyal temporal i s'ha passat les mostres a Banda Base amb unes funcions internes de la FPGA de la USRP, ja es pot començar a descodificar tot el senyal.

Primerament les dades temporals s'introdueixen en un buffer circular que ens proporciona llibertat per analitzar mostres passades sense necessitat de perdre-les. Posteriorment la lectura del buffer dependrà d'un punter que ens indica a partir de quina mostra es vol començar a llegir. El número total de mostres que llegeix serà sempre del nombre de mostres temporals associades al BW màxim, en aquest cas de 15360 mostres.

Com inicialment no s'està sincronitzat i no es té cap tipus d'informació del senyal que es rep, el sistema filtra el senyal entrant amb un filtre pass-baix que manté les 72 subportadores centrals i atenua la resta.

El filtre pass-baix es pot crear en MATLAB amb el programa 'fdatool'. En aquest s'ha creat un filtre de 128 coeficients amb una caiguda de 3dB a la freqüència 465KHz. Aquesta freqüència es correspon a la subportadora 31, al tenir el senyal entre 0 i fs/2. El LPF aplica una atenuació de 25dB en les freqüències superiors a 465KHz donant com a vàlides les 72 centrals de la trama LTE.

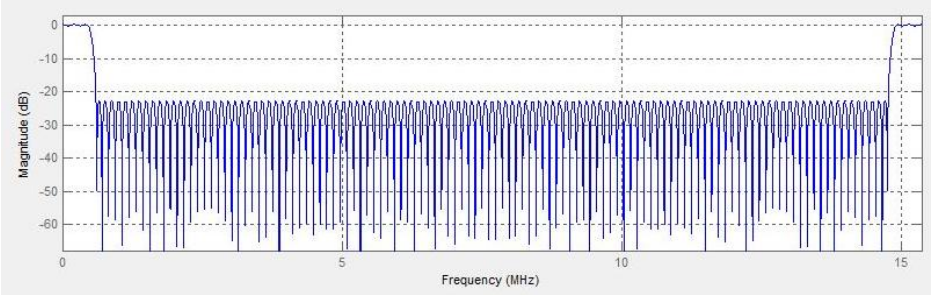


Fig 3. 12 Espectre del filtre

Per aplicar el filtre es fa una convolució d'ell mateix sobre el senyal temporal deixant l'espectre resultant següent:

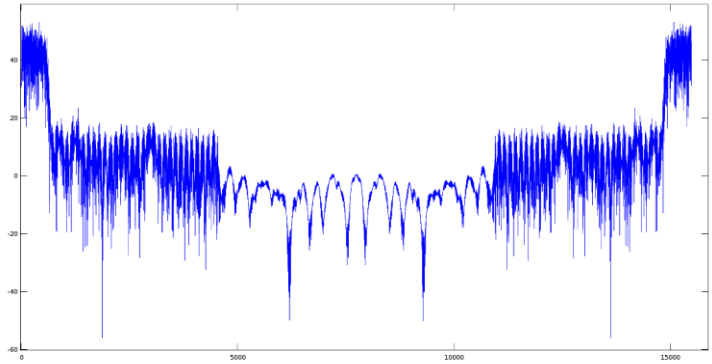


Fig 3. 13 Espectre resultant després de passar pel filtre

Com es veu a la imatge anterior, el filtre que s'ha introduït és molt bo per filtrar les subportadores superior a les 72 centrals, però per contra, ens introdueix un cost computacional molt elevat. Per tal de reduir aquest desavantatge, el que es fa és un delmat especial que s'aplica el filtre cada 8 mostres del senyal. Aquest procediment es repeteix fins a obtenir el MIB del PBCH, quedant un senyal temporal de 1920 mostres, que es corresponen a un senyal amb BW central de 1.4MHz.

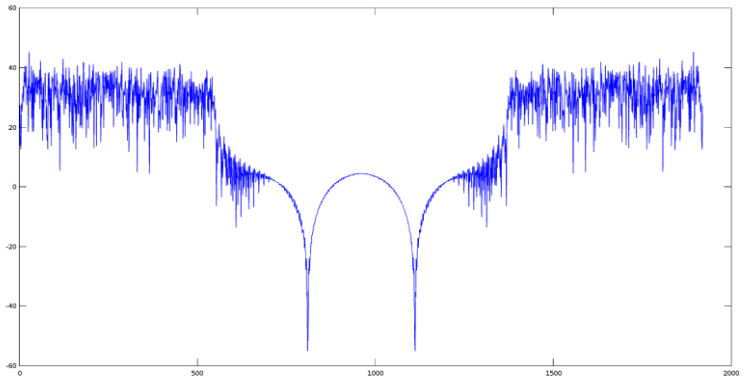


Fig 3. 14 Espectre resultant de Filtre delmador

Una vegada que el sistema té les 1920 mostres corresponents al BW de 1.4Mhz central, s'ha d'alinear el senyal rebut tant en temps com en freqüència. En el cas de ser les primeres recepcions, el senyal no estarà alineat i per tant haurà de buscar la PSS en cada recepció. En el moment en que es detecti un pic de correlació, aquest tindrà una forma semblant a la següent imatge:

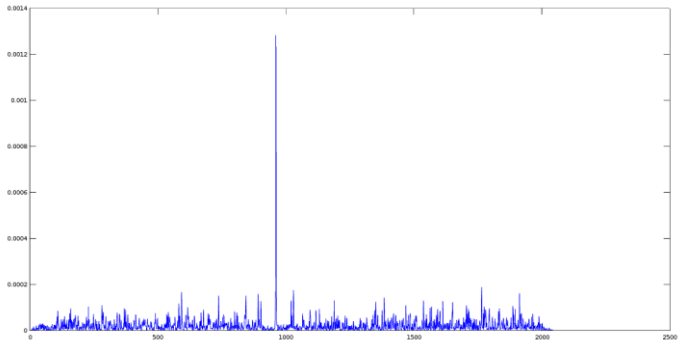


Fig 3. 15 Correlació de la PSS en mode FIND

Quan el pic de correlació sigui superior al Threshold prèviament designat, s'aplicarà un procés d'alineació de les mostres. Aquest procés calcula la mostra en que s'ha de començar a llegir del buffer circular per aconseguir que el pic de correlació quedi a la mostra central (núm. 960). En cas de produir-se l'alineació del senyal, es tindrà un retard a la recepció de les pròximes mostres en funció de la posició del pic.

Un cop alineades les mostres a la recepció, es pot trobar els pics de correlació en mode Tracking. Aquest mode, a diferència de l'anterior, fa la correlació a les mostres on està situada la PSS i redueix molt els temps en termes de computació. Un cop detectada la PSS s'extreu el valor N_id_2 i es descodifica la SSS i es pot extreure una sèrie de valors que determinen les característiques de la cèl·lula. Aquests valors són els següents: el N_id_1, el Cell_ID, aplicant la fórmula **(2.1)**; el número del subframe de la trama; el prefix cíclic associat al senyal i el tipus d'estructura (TDD o FDD).

Un cop sincronitzat i alineat el senyal, es pot descodificar el canal de broadcast PBCH i extreure la configuració total del senyal LTE. Amb aquest canal de la capa física es pot saber quin és l'Ample de Banda i quants RB utilitza, el nombre d'antenes en transmissió i el frame en que s'està transmetent. Amb el BW, es pot adaptar el senyal i determinar el valor del delmador.

En el moment en que un senyal és mostrejat en recepció amb una f_s superior a la que emet el transmissor, el senyal rebut té més mostres de les que hauria de tenir, i per tant no es pot processar correctament. El gran avantatge de LTE és que totes les f_s són múltiples i submúltiples entre elles, per tant, si es vol reduir la fs d'un senyal ja mostrejat se li pot aplicar un delmat per reduir la f_s i adaptar-la a l'Ample de banda desitjat.

En el cas que s'està implementant, la recepció de les mostres es realitza amb una f_s de 16.66Msps (recordar que s'han canviat les f_s per les USRP) que es correspon amb un BW de 10MHz. Però una vegada extret el MIB de les portadores centrals, es determina que l'Ample de Banda del senyal és de 5MHz i posseeix una f_s de 8.33Msps. Per tal d'adaptar el senyal rebut i rebre les mostres transmeses s'ha fet un delmat de 2 (16.66 Msps /8.33 Msps).

Seguint les passes descrites en el diagrama de flux Fig 3.10 la implementació, el procés de delmat en cada subframe es realitza després de rebre les dades, ficar-les al buffer circular i llegir el buffer a partir del punter de lectura.

Un cop delmat el senyal ja es disposa de les mostres correctes. Aquestes mostres s'han de tornar a sincronitzar per determinar la posició exacte del sincronisme. En l'anterior correlació s'utilitzava l'ample de banda de 1.4MHz central, i per tant, no es podia determinar exactament el pic de correlació perquè podria tenir mostres d'incertesa degut al filtre delmador. Amb el delmat agafem 1 mostra de cada N i conseqüentment no es pot saber amb exactitud si el sincronisme està en una de les mostres delmades o en la mostra no delmada.

La sincronització de la PSS segueix en mode Tracking ja que la correlació de la PSS quan s'estava filtrant donava una pista d'on podria estar el pic. Una vegada se sap el pic de correlació i la posició del senyal, s'ha de determinar si està alineada o no amb el mateix procés del buffer circular.

Un cop alineat el senyal temporal, el sistema torna a buscar el PSS per determinar que s'ha fet bé l'alineació i seguidament descodifica el SSS per determinar quin subframe està rebent l'UE. En el cas de que s'estigui en el subframe 0, el sistema descodificarà el canal PBCH seguint el mateix procés descrit en **l'apartat 2.3.1.2**. Aquest comprovarà que els paràmetres del MIB són els correctes i determinarà que el sistema està totalment sincronitzat per descodificar les dades. En cas que sigui el subframe 5, simplement descodificarà els senyals de sincronisme.

En el cas que s'hagi determinat el sincronisme total del sistema, serà l'hora de descodificar els canals físics PCFICH, PDCCH i PDSCH. La descodificació seguirà les passes descrites en els **apartats 2.4.1.2, 2.4.2.2 i 2.3.2.2**, successivament.

3.2.1. UE en ALOE

En aquest apartat l'objectiu és fer funcionar tot el sistema receptor explicat anteriorment i fer-lo simular com si fos un UE capaç de rebre les dades enviades per l'eNodeB.

En ALOE cada mòdul està dividit en un procés d'inicialització de les variables i un procés d'execució i processat del senyal que li arriba d'altres mòduls. Per tant, ara s'ha de donar un cop d'ull a la implementació anterior i separar la part d'inicialització de les variables de la part de processament del senyal.

Una vegada s'ha diferenciat els dos processos dins de la implementació de l'UE, s'ha d'observar els temps d'execució de cada part al programa anterior. Els temps més costosos són les inicialitzacions intermitges que es produeixen depenen del moment del programa. Altres funcions que tenen un temps d'execució alt són les funcions que tenen operacions molt pesades com aplicar filtres, fer la FFT i les convolucions amb moltes mostres, el càlcul de l'estimació de canal, i l'aplicació els decoders i unrate matchings. Hi ha d'altres operacions menys costes, però si agrupem moltes d'aquestes, poden provocar un cost significatiu. Aquestes operacions són les divisions i els mòduls.

Depenent del tipus de PC en que es facin les simulacions, els temps de processat variarà de forma considerable. Un exemple el trobem en la comparació dels temps d'execució d'un PC amb un processador Intel Core Dos Duo amb un Intel I7 Quad Core, ambdós ordinadors amb una freqüència 2.2GHz. Aquest exemple està fet pel senyal amb un BW de 1.4MHz.

	Temps processat Intel Core Dos Dus (ms)	**Temps processat Intel I7 Quad Core (ms)**
Filtratge a BW 1.4MHz	0.85	0.55
Delmat	0.023	0.013
Alineació de la trama	0.01	0.005
Execució PSS mode FIND	0.443	0.19
Execució PSS mode TRACK	0.079	0.03
Descodificació SSS	0.04	0.03
Descodificació PBCH	0.2	0.15
Inicialització objectes PBCH	0.9	0.45
Descodificació canals PCFIC, PDCCH i PDSCH	0.3	0.15
Inicialització objectes canals PCFIC, PDCCH i PDSCH	2.1	0.75

Taula 3. 2 Temps processat de l'UE en dos processadors

Amb el diagrama de blocs de la figura **Fig. 3.10** es pot comprovar que si utilitzem un Intel Core Dos Duo, els temps de processat total seran superiors al temps de TTI, mentre que si fem el mateix procediment amb un Intel I7 els temps totals de processament sempre estaran per sota de 1 ms. Aquí queda demostrar la importància de tenir un bon processador en aquest tipus d'aplicacions.

Els mòduls d'ALOE que s'han creat per rebre el senyal LTE són tres i són els següents:

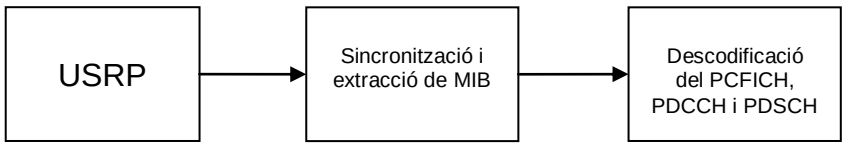


Fig 3. 16 Procés d'execució de l'UE en ALOE

En el mòdul de la USRP, la pròpia USRP utilitza la FPGA per utilitzar el Digital Down Convert (DDC) on es porta el senyal que està a la freqüència portadora f_o a Banda Base centrada en 0. I seguidament aplica l'ADC per passar el senyal analògic a discret i tenir el senyal mostrejat.

En el següent mòdul 'Sincronització i extracció del MIB' entren les 15.360 mostres corresponents a 10MHz d'ample de banda. Aquestes mostres passen per tot el procés de sincronització i extracció del MIB explicat anteriorment en la implementació del UE. Les mostres resultants d'aquest bloc són el senyal temporal delmat i alineat tant en temps com en freqüència, per tal que sigui més senzill descodificar-la.

En l'últim mòdul anomenat 'Descodificació del PCFICH, PDCCH i PDSCH' s'obtenen els bits d'informació que envia l'eNodeB. Segons la implementació del UE explicada anteriorment, el mòdul es correspon al moment en que el sistema està totalment sincronitzat. Aquest mòdul és per un únic UE amb un determinat RNTI. En el cas de tenir més d'un UE es repetiria múltiples vegades amb el mateix senyal d'entrada.

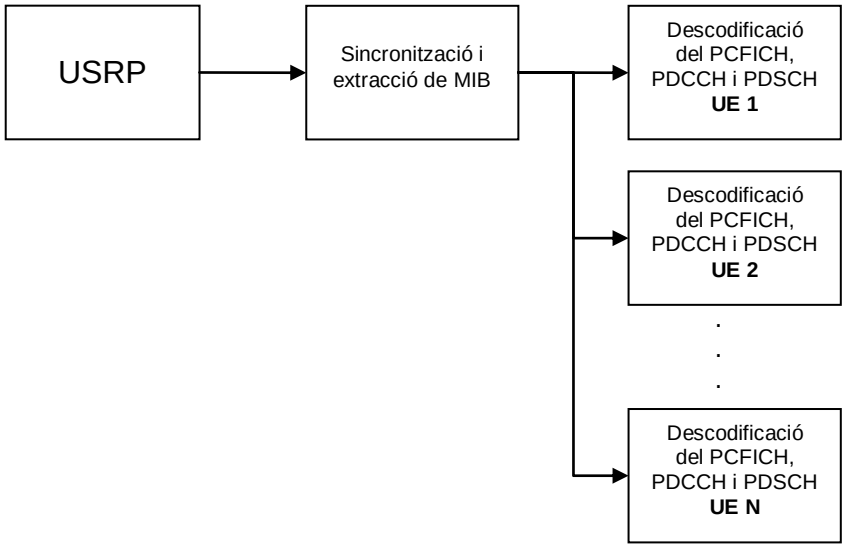


Fig 3. 17 Procés d'execució en recepció amb multi UEs en ALOE

CAPÍTOL 4. RESULTATS

En aquest capítol veurem quins són els resultats que s'han obtingut de la implementació tant de l'eNodeB com del UE. Per estructurar la informació s'ha dividit el capítol en dues parts, una part referenciada a l'eNodeB i l'altra part està referenciada al UE.

4.1. Implementació eNode sense ALOE

Els resultats de l' implementació de l'eNodeB han estat molt satisfactoris, ja que s'ha aconseguit generar el senyal LTE en l'enllaç Downlink en tots els Amples de Banda que fa referència l'estàndard LTE en 3GPP. S'han introduït satisfactòriament tots els canals de la capa física, els senyals sincronisme i els de referència, en les posicions adients.

Com s'ha explicat en la implementació de l'eNodeB en el l'**apartat 3.1**, s'ha introduït dos modes de transmissió del senyal. El primer mode es basa a simular un canal AWGN per on passa el senyal generat, seguidament es copien les dades resultants en un arxiu per tal de ser llegides a posteriori pel receptor. El segon mode es basa a enviar el senyal generat per una USRP a través del canal real.

4.1.1. eNodeB amb un canal AWGN

En el primer cas s'ha generat el senyal amb una Ample de banda de 1.4MHz i una relació SNR de -3dB, 2dB, 7dB i 17 dB. Per determinar els efectes del canal gaussià s'ha de veure l'espectre del senyal, per fer-ho s'ha utilitzat el programa 'OCTAVE'.

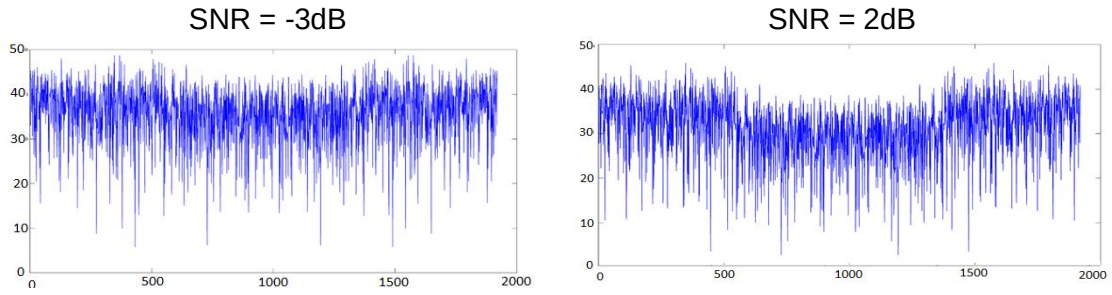


Fig 4. 1 Espectres del senyal (BW=1.4MHz) amb diferents relacions SNR

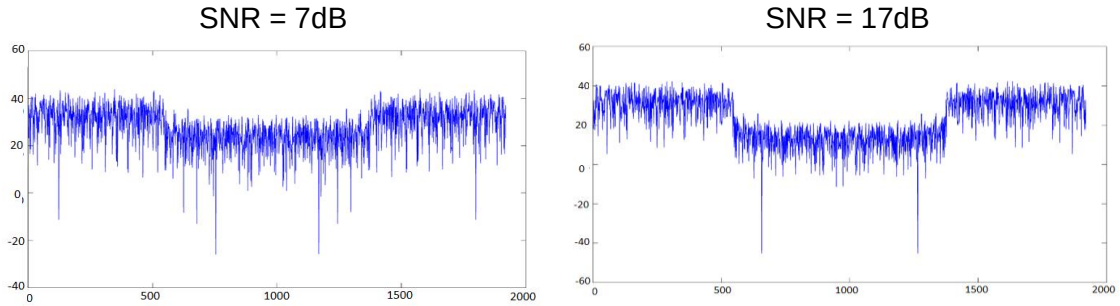


Fig 4. 2 Espectres del senyal (BW=1.4MHz) amb diferents relacions SNR

S'ha fet una segona prova per demostrar el correcte funcionament del eNodeB amb un canal AWGN. En aquesta segona prova s'ha generat els restants amples de banda amb una relació SNR de 27dB.

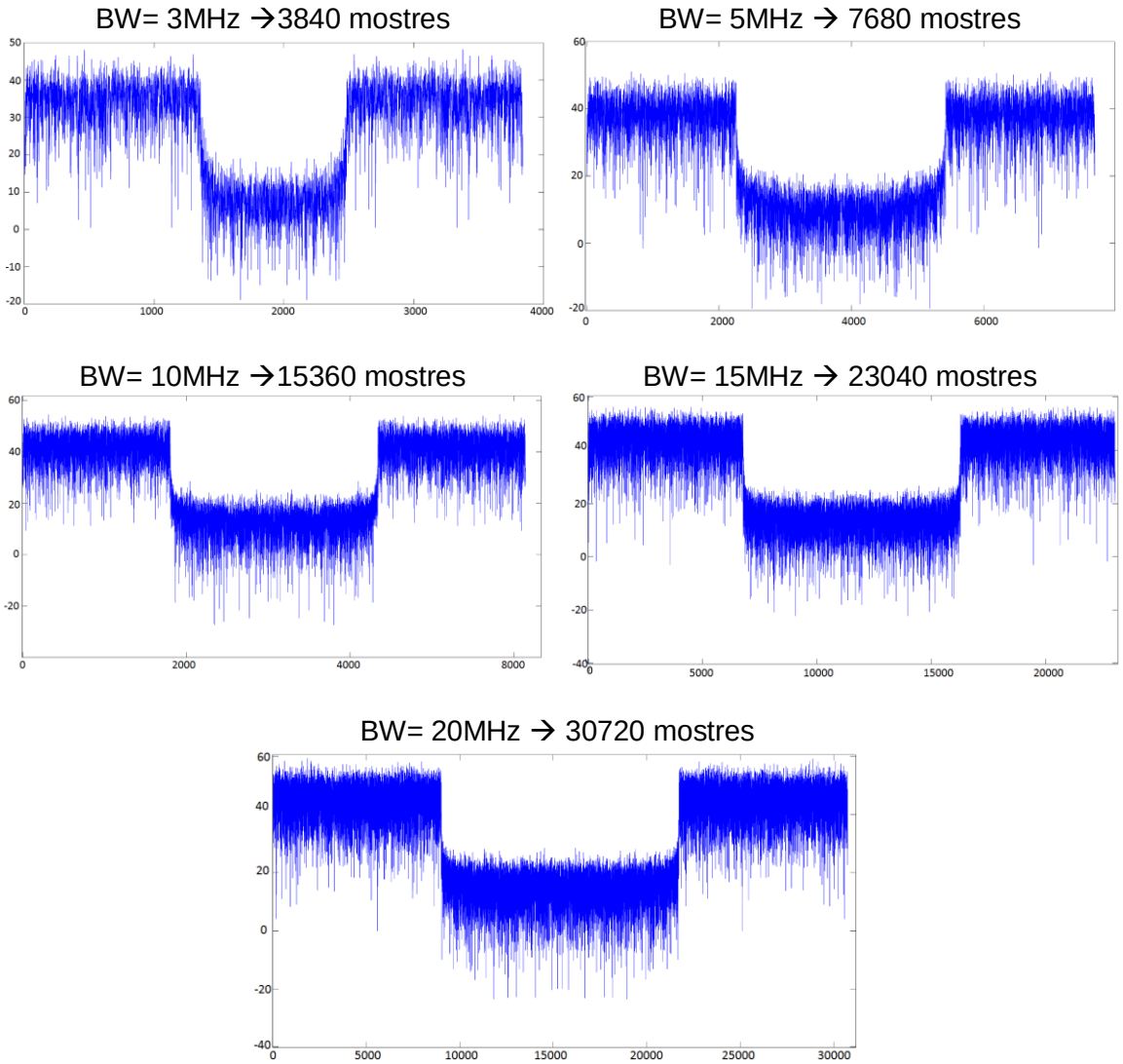


Fig 4. 3 Espectre dels senyals amb diferents amples de banda.

4.1.2. eNodeB amb un canal real

En aquest segon cas, l'eNodeB té connectada una USRP que envia el senyal a través del canal real. Per comprovar el correcte funcionament de la transmissió, el que s'ha fet és intentar enviar diferents senyals amb diferents Amples de banda de LTE i rebre el senyal a través d'una segona USRP connectada a un PC.

Gràcies al GNU Radio, hi ha una aplicació anomenada 'uhd_fft' que captura els senyals que rep la USRP entre les freqüències 2GHz i 2.9GHz. Per poder aconseguir la detecció del senyal generat per l'eNodeB, aquest s'envia a través de la freqüència portadora 2.39GHz. Com el senyal està dintre del rang de les freqüències de la USRP, es pot veure l'espectre capturat per ella.

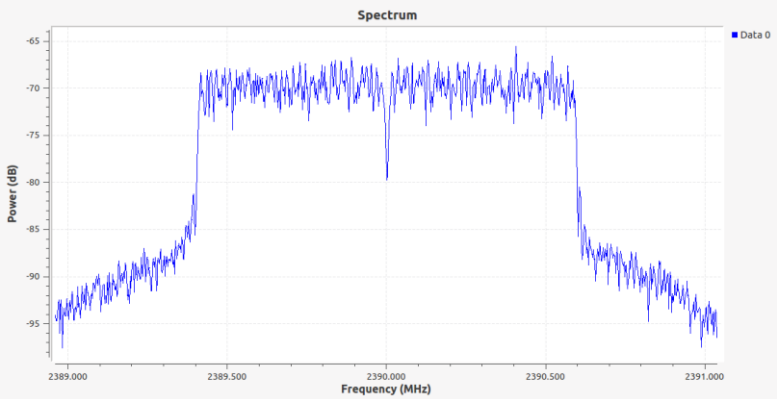


Fig 4. 4. Espectre amb un BW= 1.4MHz i una SNR de 15dB

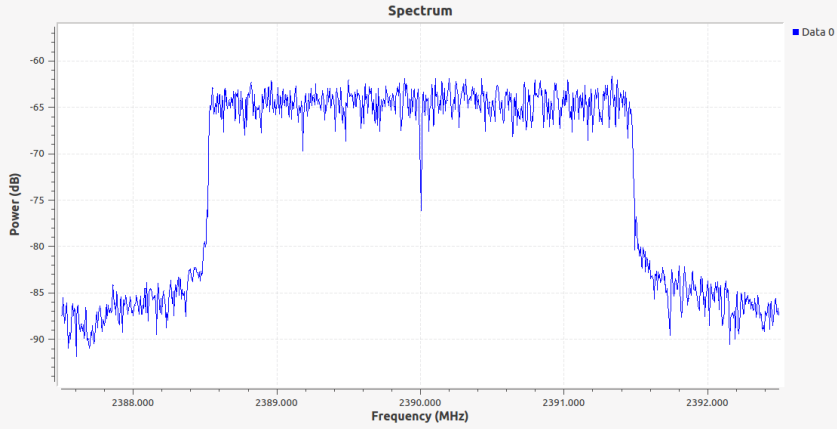


Fig 4. 5 Espectre amb un BW=3MHz i una SNR de 18dB

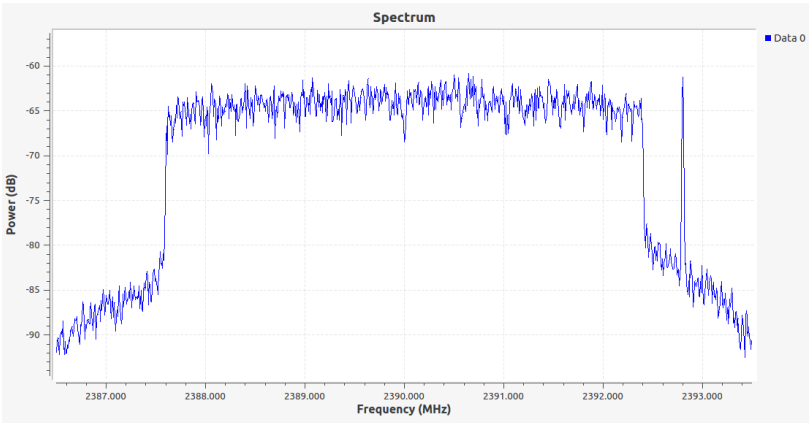


Fig 4. 6 Espectre amb un BW= 5MHz, SNR de 18dB i un pic d'interferència

Com podem observar, les transmissions en els casos de 1.4MHz, 3 MHz i 5 MHz han resultat satisfactoris. En el demes casos no es va poder cap senyal LTE perquè la USRP donava problemes per poder enviar el senyal. Aquests problemes són deguts a fet que el processament del cada subframe supera el temps de TTI (1ms).

Els resultats obtinguts concorden amb **Taula 3.1** on s'indiquen quins són els temps de processament del subframes amb els diferents amples de banda, i quins es passen del temps de TTI.

Una solució per poder enviar senyals LTE amb un alt BW és transmetre aquest senyal amb super-ordenadores, capaços de suportat una gran computació en poc temps.

4.2. Implementació eNodeB amb ALOE

Igual que en l'apartat anterior l'eNodeB genera perfectament el senyal LTE en el amples de banda de 1.4MHz, 3MHz i 5MHz. En els altres, ALOE ens indica que hi ha problemes de Real Time.

Els problemes de Real Time s'esdevenen en el moment que els tres mòduls implementats s'executen en més d'un Time Slot. Cadascun d'aquests estan configurats en 1ms per tal de complir amb els requisits del TTI marcats per l'estàndard.

4.3. Implementació UE sense ALOE

Com s'ha explicat en la implementació del UE en **l'apartat 3.2,** s'ha introduït dos modes de recepció del senyal. El primer mode està basat en la lectura d'un arxiu de dades, les quals es corresponen a una senyal LTE a través d'un canal AWGN. El segon mode es basa en rebre el senyal generat a través d'una USRP.

4.3.1 UE a través d'un canal AWGN

En aquest primer cas, el senyal es rep a través d'un fitxer de dades, on aquestes dades indiquen com és la senyal temporal després d'haver passat per un canal AWGN.

Com hem explicat en la implementació del UE, el sistema està pensat per rebre senyals LTE amb BW de 10MHz com a màxim. Això és degut a que el nostre sistema ja està encarat a rebre el senyal des de la USRP i processar-l'ho per tal d'extreure la informació destinada al usuari.

La primera prova que s'ha fet és la descodificació d'un senyal amb un BW de 1.4MHz, un TBS de 152 bits, una modulació QPSK del canal PHY PDSCH, i el turbo coder que apliquem ens permet fins a 3 iteracions. Els resultats són els següents:

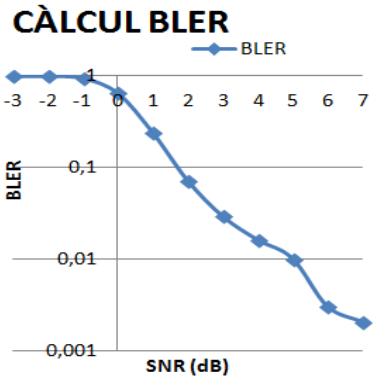


Fig 4. 7 BLER vs SNR

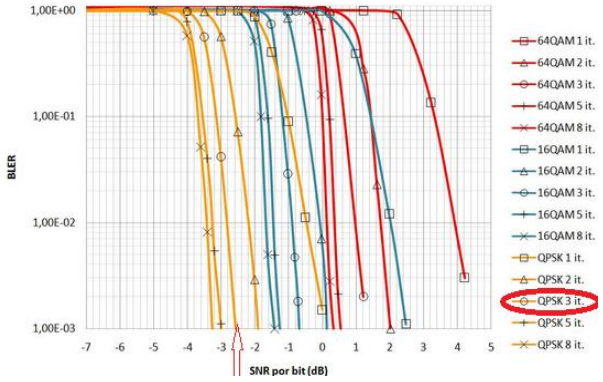


Fig 4. 8 BLER vs SNR en el sistema IDEAL [16]

La BLER que ens determina la implementació del receptor en comparació amb la BLER d'un sistema real ens indica que el moment de la caiguda està desplaçat 3dB i el pendent no és l'adequat. Això pot ser degut al fet que el turbo codi i el rate matching no estiguin implementats correctament i hi hagi errors en el codi.

A continuació es mostraran quatre gràfiques diferents coa a resultat d'analitzar el Unrate Matching turbo. En la primera gràfica el Unrate matching no actua perquè té un marge del 5% de bits, en la segona gràfica el Unrate Matching si que actua i ha de reduir en un 5% la mida de bits de l'entrada. En la tercera gràfica, el Unrate Matching actua per reduir un 25% dels bits d'entrada. I per últim, la quarta gràfica analitza el comportament del Unrate Matching quan actua per reduir un 55% dels bits d'entrada.

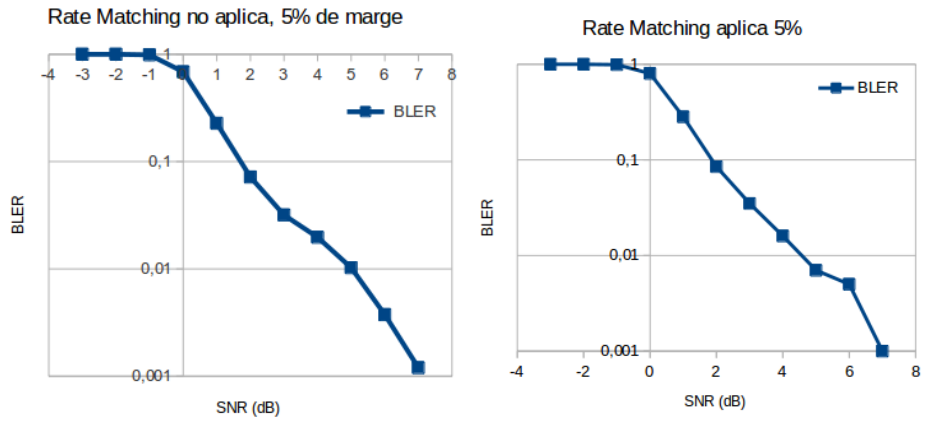


Fig 4. 9. BLER vs SNR rate maching (1)

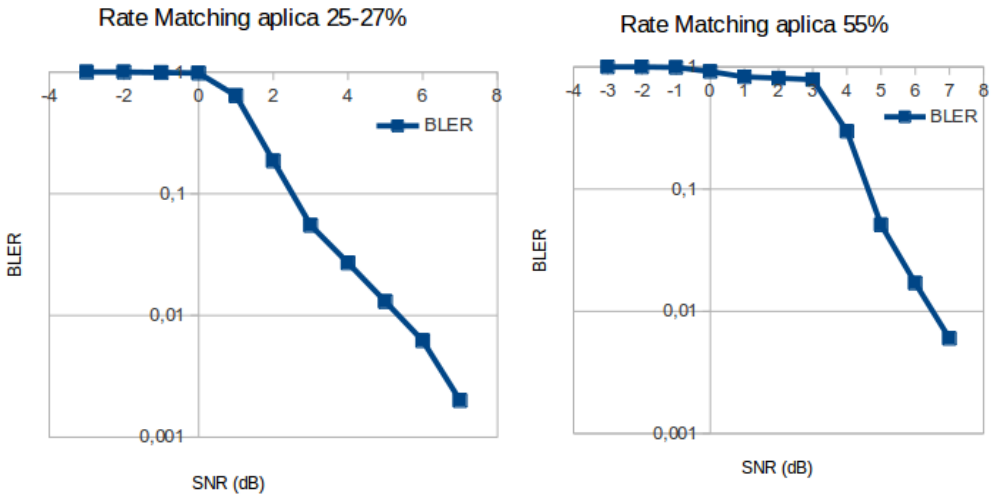


Fig 4. 10. BLER vs SNR rate maching (2)

En les gràfiques es pot apreciar quan s'aplica el Unrate matching en pocs bits, el rendiment d'aquest és igual que quan no s'aplica. A mesura que el nombre de bits, en que es vulgui aplicar el Unrate matching, va augmentant, el rendiment d'aquest va disminuint de manera progressiva. Un exemple clar el trobem en els valors de la gràfica, ja que Unrate matching que s'aplica un 55% dels bits necessita una SNR 3 dB més gran per aconseguir una BLER≤0.1.

4.3.2. UE a través de la USRP

En les proves que s'han fet amb la USRP, s'han donat molts problemes amb la part de Unrate Matching. A diferència del cas anterior, quan arriba el moment que s'ha d'aplicar el unrate matching al procés de descodificació del PDSCH, aquest no s'executa de forma correcta i no pot descodificar les dades d'informació. En la gran majoria dels casos, quan no s'aplica el Unrate matching la implementació funciona perfectament. Aquí tenim unes taules que ens mostren quines parts de la implementació funciona i en quin moment ho fan.

BW	1,4MHz									
#TBS	0	1	2	3	4	5	6	7	8	9
bits TBS	152	208	256	328	408	504	600	712	808	936
SF_0										
SF_1										
SF_5										
BLER	0,101833	0,102642	0,219388	0,204893	0,32719	1	1	1	1	1
Modulació	QPSK									
bits SF 0	675									
bits SF 1	1512									
bits SF 5	1224									
BITS CODIFICATS	540	708	852	1068	1308	1596	1884	2220	2508	2892
RM SF_0	NO	SI	SI	SI	SI	SI	SI	SI	SI	SI
RM SF_1	NO	NO	NO	NO	NO	SI	SI	SI	SI	SI
RM SF_5	NO	NO	NO	NO	SI	SI	SI	SI	SI	SI

Fig 4. 11 Informació del senyal amb BW=1.4MHz

BW	1,4MHz									
#TBS	0	1	2	3	4	5	6	7	8	9
bits TBS	152	208	256	328	408	504	600	712	808	936
SF_0										
SF_1										
SF_5										
BLER	0,101833	0,102642	0,219388	0,204893	0,32719	1	1	1	1	1
Modulació	QPSK									
bits SF 0	675									
bits SF 1	1512									
bits SF 5	1224									
BITS CODIFICATS	540	708	852	1068	1308	1596	1884	2220	2508	2892
RM SF_0	NO	SI	SI	SI	SI	SI	SI	SI	SI	SI
RM SF_1	NO	NO	NO	NO	NO	SI	SI	SI	SI	SI
RM SF_5	NO	NO	NO	NO	SI	SI	SI	SI	SI	SI

Fig 4. 12 Informació del senyal amb BW=3MHz

BW	5MHz									
#TBS	0	1	2	3	4	5	6	7	8	9
bits TBS	680	904	1096	1416	1800	2216	2600	3112	3496	4008
SF_0										
SF_1										
SF_5										
BLER	0,001	0,005325	0,008155	0,027636	0,12349	1	1	1	1	1
Modulació	QPSK									
bits SF 0	6060									
bits SF 1	6611									
bits SF 5	6900									
BITS CODIFICATS	2124	2796	3372	4332	5484	6732	7884	9420	10572	12108
RM SF_0	NO	NO	NO	NO	NO	SI	SI	SI	SI	SI
RM SF_1	NO	NO	NO	NO	NO	SI	SI	SI	SI	SI
RM SF_5	NO	NO	NO	NO	NO	NO	SI	SI	SI	SI

Fig 4. 13 Informació del senyal amb BW=5MHz

En aquestes taules podem observar tota la informació referent al sistema quan està rebent amb la USRP. Els valors de les taules estan determinats pel TBS que s'està descodificant en aquell moment. Les files corresponents a SF_0, SF_1 i SF_5 indiquen l'estat del subframe referent a la descodificació del PDSCH; en color verd indica que es descodifica correctament, en vermell indica que no es descodifica en cap cas i en groc indica que es descodifica depenent de la transmissió. Cal especificar que el subframe 'SF_1' fa

referència a tots els subframes de la trama que són igual a ell, és a dir, tots excepte el 0 i el 5.

En tots els casos es marca la BLER calculada pel sistema, gràcies aquest valor es pot determinar que tenim un problema solament amb el Unrate matching ja que deixa de descodificar quan aquest s'aplica en el procés de descodificació.

4.4. Implementació UE sobre ALOE

Igual en els apartats anteriors el receptor funciona igual en els dos modes de recepció del senyal. En el mode en que es llegeix el senyal amb un soroll AWGN des d'un un fitxer de dades, la BLER pateix els mateixos efectes en funció de la SNR. En el cas en que el mode de propagació és un canal real i es rep mitjançant una USRP, els senyals LTE amb els Amples de Banda de 3 Mhz i 5MHz funcionen correctament. En el cas del senyal amb un BW de 5MHz s'ha tingut petits problemes de Real Time que feien variar la BLER.

CAPÍTOL 5. CONCLUSIONS

Com s'ha vist durant tot el projecte, la implementació d'un eNodeB i d'un UE en LTE és molt complicat. Primerament s'ha de tenir uns amplis coneixements de l'estructura de la capa física de LTE, i seguidament s'ha de tenir un gran coneixement del procés d'implementació que segueixen cadascun dels elements en l'enllaç descendent. També s'ha de saber totes les possibles casuístiques per prevenir errors, i en cas de tenir-los s'ha de saber com resoldre'ls.

En relació a la llibreria que hem utilitzat per implementar ambdós processos, hem arribat a la següent conclusió: s'ha d'ajustar els elements Rate Matching i Turbo coder, ja que poden tenir petits errors en la seva implementació. Per tal de fer-ho s'ha d'acotar els errors i intentar solucionar-los. En una primera instancia, es tenia la idea que la llibreria podria contenir desajustaments en ser un codi OpenSource, i en finalitzar el projecte s'ha pogut comprovar que el raonament era correcte. A part dels elements Rate Matching i Turbo coder, no s'ha tingut cap problema important.

I per últim, un punt molt important que s'ha tingut molt present al llarg de la implementació és el temps de processat. Aquest és molt important aplicar-lo en cada procés per tal de tenir una referència per calcular el temps total. Per tal d'executar qualsevol de les dues implementacions (eNodeB i UE) cal tenir un ordinador bastant potent en termes de computació, aquest farà que els temps es redueixin de manera molt significativa.

CAPÍTOL 6. TREBALL DE FUTUR

Com es mostren els resultats obtinguts en la implementació ens obren molts camins per seguir desenvolupant la tecnologia LTE.

El primer punt que s'hauria de resoldre seria revisar les funcions que implementen el Rate Matching i el Turbocodi, tant la part de codificació com la part de descodificació, per veure quins són els motius pels quals no es descodifiquen les dades correctament en el canal de la capa física PDSCH.

Un altre punt molt important a resoldre seria reduir temps de processatment de la implementació tant en l'eNodeB com en la UE. Per tal d'aconseguir aquesta fita, s'hauria de revisar el codi C i millorar-lo de tal forma que si en algun moment es vol introduir en ALOE no s'hauria de tenir problemes de Real-Time. A més a més ajudaria a transmetre els senyals LTE amb amples de banda grans, ja que tenen un alt cost computacional.

El funcionament de la USRP ens ha fet retocar paràmetres de configuració de LTE, això ha fet que el nostre senyal no compleixi 100% les especificacions marcades a l'estàndard del 3GPP. Per aquest motiu, una altra millora possible seria la d'incorporar un "resampler" per poder ajustar la freqüència de mostreig a les exigides per l'estàndard LTE.

Un altre aspecte important seria implementar l'enllaç Uplink per tal el que UE pugui enviar informació a l'eNodeB i incorporar el HARQ que ha de permetre millorar la qualitat de la comunicació.

I per últim, es podria implementar tota la part de la capa MAC de la tecnologia LTE.

BIBLIOGRAFÍA

[1] http://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf

[2] Gelonch, Antoni; Software Radio Engineering_Corsev3[pdf], Gelonch Antoni, 2013

[3] 3GPP. 3GPP TS 36.**211** V8.3.0 (2009-04) Physical Channels and Modulation (Release8). Technical report, 3GPP, December 2009. **Apartat 6.11.1.**

[4] 3GPP. 3GPP TS 36.**211** V8.3.0 (2009-04) Physical Channels and Modulation (Release8). Technical report, 3GPP, December 2009. **Taula 6.11.2.1-1**

[5] 3GPP. 3GPP TS 36.**211** V8.3.0 (2009-04) Physical Channels and Modulation (Release8). Technical report, 3GPP, December 2009. **Apartat 6.10.1.1**

[6] 3GPP. 3GPP TS 36.**212** V8.3.0 (2009-04) Physical Channels and Modulation (Release8). Technical report, 3GPP, December 2009. **Apartat 5.1.3.1**

[7] García Lozano, Mario; LTE Long Term Evolution of 3GPP[pdf], 2015.

[8] 3GPP. 3GPP TS 36.**213** V8.3.0 (2009-04) Physical Channels and Modulation (Release8). Technical report, 3GPP, December 2009. **Taula 7.1.7.2.1-1**

[9] 3GPP. 3GPP TS 36.**212** V8.3.0 (2009-04) Physical Channels and Modulation (Release8). Technical report, 3GPP, December 2009. **Apartat 5.1.2**

[10] 3GPP. 3GPP TS 36.**212** V8.3.0 (2009-04) Physical Channels and Modulation (Release8). Technical report, 3GPP, December 2009. **Apartat 5.1.3.2**

[11] 3GPP. 3GPP TS 36.**212** V8.3.0 (2009-04) Physical Channels and Modulation (Release8). Technical report, 3GPP, December 2009. **Apartat 5.1.5**

[12] 3GPP. 3GPP TS 36.**212** V8.3.0 (2009-04) Physical Channels and Modulation (Release8). Technical report, 3GPP, December 2009. **Apartat 5.3.4.1**

[13] 3GPP. 3GPP TS 36.**212** V8.3.0 (2009-04) Physical Channels and Modulation (Release8). Technical report, 3GPP, December 2009. **Apartat 5.3.3**

[14] Lozano , Mario; 3GPP LTE: HACIA LA 4G MÓVIL, S.A MARCOMBO, 2012

[15] http://dhagle.in/LTE

[16] http://dtstc.ugr.es/tl/research_broadband.html

ANNEXES

ANNEX 1: Trames LTE

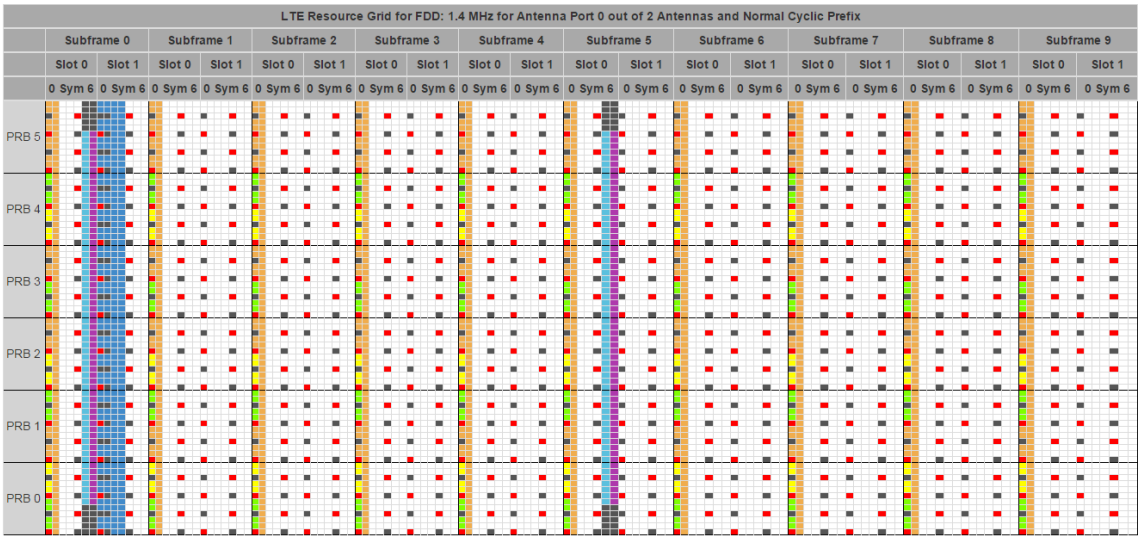


Fig A1. 1 Trama BW = 1.4MHz [15]

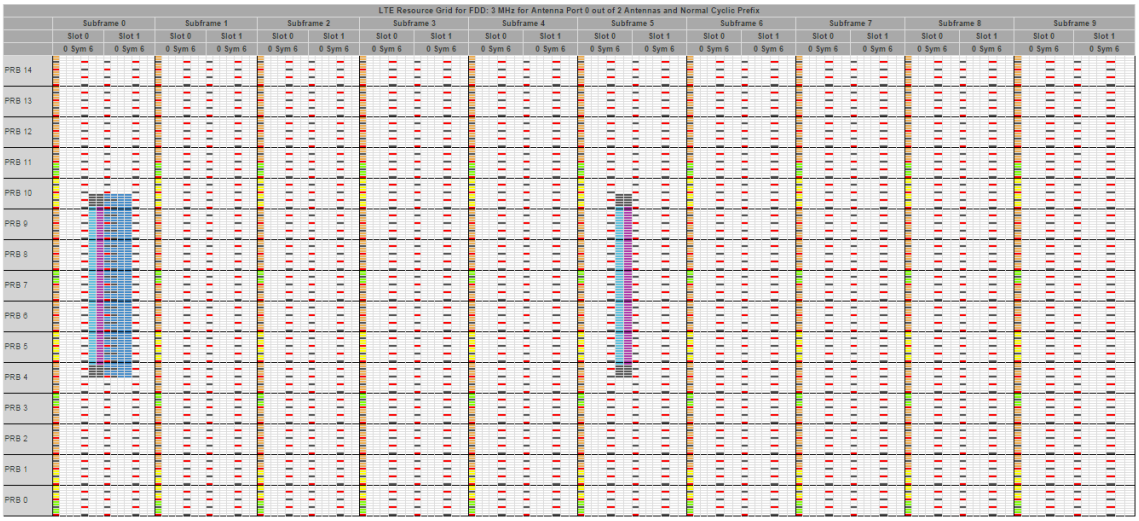


Fig A1. 2 Trama BW = 3MHz [15]

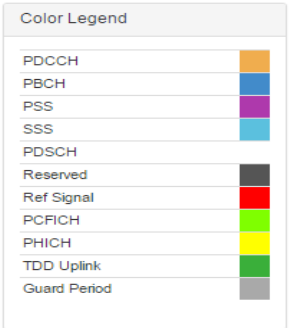


Fig A1. 3 Llegenda dels canals LTE

ANNEX 2: CODI DE LA IMPLEMENTACIÓ DE L'ENODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <unistd.h>
#include <sys/select.h>
#include <pthread.h>
#include <semaphore.h>

#include "srslte/srslte.h"

#ifndef DISABLE_UHD
#include "srslte/cuhd/cuhd.h"
void *uhd;
#endif

char *output_file_name = NULL;
char *input_file_name = NULL;

#define LEFT_KEY  68
#define RIGHT_KEY 67
#define UP_KEY    65
#define DOWN_KEY  66

srslte_cell_t cell = {
  6,            // nof_prb
  1,            // nof_ports
  0,            // bw idx
  0,            // cell_id
  SRSLTE_CP_NORM,       // cyclic prefix
  SRSLTE_PHICH_R_1,          // PHICH resources
  SRSLTE_PHICH_NORM    // PHICH length
};

int kk=1;

int net_port = -1; // -1 generates random data

uint32_t cfi=2;
uint32_t mcs_idx = 1, last_mcs_idx = 1, mcs_idx_system=1;
int nof_frames = -1;

char *uhd_args = "";
float uhd_amp = 0.8, uhd_gain = 70.0, uhd_freq = 2400000000;

bool null_file_sink=false;
srslte_filesink_t fsink;
srslte_filesink_t fsink2;
srslte_filesource_t fsource;
srslte_ofdm_t ifft;
srslte_pbch_t pbch;
```

```
srslte_pcfich_t pcfich;
srslte_pdcch_t pdcch;
srslte_pdsch_t pdsch;
srslte_pdsch_cfg_t pdsch_cfg;
srslte_softbuffer_tx_t softbuffer;
srslte_regs_t regs;
srslte_ra_dl_dci_t ra_dl;

float SNR=0.0;

cf_t *sf_buffer = NULL, *output_buffer = NULL;
int sf_n_re, sf_n_samples;

int prbset_num = 1, last_prbset_num = 1;
int prbset_orig = 0;

void usage(char *prog) {
  printf("Usage: %s [agmfoncvpuV]\n", prog);
#ifndef DISABLE_UHD
  printf("\t-a UHD args [Default %s]\n", uhd_args);
  printf("\t-l UHD amplitude [Default %.2f]\n", uhd_amp);
  printf("\t-g UHD TX gain [Default %.2f dB]\n", uhd_gain);
  printf("\t-f UHD TX frequency [Default %.1f MHz]\n", uhd_freq / 1000000);
#else
  printf("\t   UHD is disabled. CUHD library not available\n");
#endif
  printf("\t-i input_file [Default 'Random bits']\n");
  printf("\t-o output_file [Default USRP]\n");
  printf("\t-m MCS index [Default %d]\n", mcs_idx);
  printf("\t-n number of frames [Default %d]\n", nof_frames);
  printf("\t-c cell id [Default %d]\n", cell.id);
  printf("\t-p nof_prb [Default %d]\n", cell.nof_prb);
  printf("\t-S SNR [Default %f]\n", SNR);
  printf("\t-v [set srslte_verbose to debug, default none]\n");
}

void parse_args(int argc, char **argv) {
  int opt;
  while ((opt = getopt(argc, argv, "aglfmioncpvuS")) != -1) {
    switch (opt) {
    case 'a':
      uhd_args = argv[optind];
      break;
    case 'g':
      uhd_gain = atof(argv[optind]);
      break;
    case 'l':
      uhd_amp = atof(argv[optind]);
      break;
    case 'f':
      uhd_freq = atof(argv[optind]);
      break;
    case 'i':
      input_file_name = argv[optind];
      break;
```

```
    case 'o':
      output_file_name = argv[optind];
      break;
    case 'm':
      mcs_idx = atoi(argv[optind]);
      break;
    case 'n':
      nof_frames = atoi(argv[optind]);
      break;
    case 'p':
      cell.nof_prb = atoi(argv[optind]);
      break;
    case 'S':
      SNR = atoi(argv[optind]);
      break;
    case 'c':
      cell.id = atoi(argv[optind]);
      break;
    case 'v':
      srslte_verbose++;
      break;
    default:
      usage(argv[0]);
      exit(-1);
    }
  }
#ifdef DISABLE_UHD
  if (!output_file_name) {
    usage(argv[0]);
    exit(-1);
  }
#endif
}

void base_init() {

  /* init memory */
  sf_buffer = malloc(sizeof(cf_t) * sf_n_re);
  if (!sf_buffer) {
    perror("malloc");
    exit(-1);
  }
  output_buffer = malloc(sizeof(cf_t) * sf_n_samples);
  if (!output_buffer) {
    perror("malloc");
    exit(-1);
  }
  if (input_file_name) {
      if (strcmp(input_file_name, "NULL")) {
        if (srslte_filesource_init(&fsource, input_file_name, SRSLTE_INT)) {
          fprintf(stderr, "Error opening file %s\n", input_file_name);
          exit(-1);
        }
    }
  }
```

```
  /* open file or USRP */
  if (srslte_filesink_init(&fsink2, "SALIDA.txt", SRSLTE_COMPLEX_FLOAT)) {
          fprintf(stderr, "Error opening file %s\n", "SALIDA.txt");
          exit(-1);
  }
  if (output_file_name) {
    if (strcmp(output_file_name, "NULL")) {
      if (srslte_filesink_init(&fsink, output_file_name, SRSLTE_COMPLEX_FLOAT)) {
        fprintf(stderr, "Error opening file %s\n", output_file_name);
        exit(-1);
      }

      null_file_sink = false;
    } else {
      null_file_sink = true;
    }
  } else {
#ifndef DISABLE_UHD
    printf("Opening UHD device...\n");
    if (cuhd_open(uhd_args, &uhd)) {
      fprintf(stderr, "Error opening uhd\n");
      exit(-1);
    }
#else
    printf("Error UHD not available. Select an output file\n");
    exit(-1);
#endif
  }

  /* create ifft object */
  if (srslte_ofdm_tx_init(&ifft, SRSLTE_CP_NORM, cell.nof_prb)) {
    fprintf(stderr, "Error creating iFFT object\n");
    exit(-1);
  }
  srslte_ofdm_set_normalize(&ifft, true);
  if (srslte_pbch_init(&pbch, cell)) {
    fprintf(stderr, "Error creating PBCH object\n");
    exit(-1);
  }

  if (srslte_regs_init(&regs, cell)) {
    fprintf(stderr, "Error initiating regs\n");
    exit(-1);
  }

  if (srslte_pcfich_init(&pcfich, &regs, cell)) {
    fprintf(stderr, "Error creating PBCH object\n");
    exit(-1);
  }

  if (srslte_regs_set_cfi(&regs, cfi)) {
    fprintf(stderr, "Error setting CFI\n");
    exit(-1);
  }
```

```
  if (srslte_pdcch_init(&pdcch, &regs, cell)) {
    fprintf(stderr, "Error creating PDCCH object\n");
    exit(-1);
  }

  if (srslte_pdsch_init(&pdsch, cell)) {
    fprintf(stderr, "Error creating PDSCH object\n");
    exit(-1);
  }

  srslte_pdsch_set_rnti(&pdsch, 1234);

  if (srslte_softbuffer_tx_init(&softbuffer, cell)) {
    fprintf(stderr, "Error initiating soft buffer\n");
    exit(-1);
  }
}

void base_free() {

  srslte_softbuffer_tx_free(&softbuffer);
  srslte_pdsch_free(&pdsch);
  srslte_pdcch_free(&pdcch);
  srslte_regs_free(&regs);
  srslte_pbch_free(&pbch);

  srslte_ofdm_tx_free(&ifft);

  if (sf_buffer) {
    free(sf_buffer);
  }
  if (output_buffer) {
    free(output_buffer);
  }
  if (input_file_name) {
      srslte_filesource_free(&fsource);
  }

  srslte_filesink_free(&fsink2);
  if (output_file_name) {
    if (!null_file_sink) {
      srslte_filesink_free(&fsink);
    }
  } else {
#ifndef DISABLE_UHD
    cuhd_close(&uhd);
#endif
  }
}

unsigned int
reverse(register unsigned int x)
{
    x = (((x & 0xaaaaaaaa) >> 1) | ((x & 0x55555555) << 1));
    x = (((x & 0xcccccccc) >> 2) | ((x & 0x33333333) << 2));
```

```
    x = (((x & 0xf0f0f0f0) >> 4) | ((x & 0x0f0f0f0f) << 4));
    x = (((x & 0xff00ff00) >> 8) | ((x & 0x00ff00ff) << 8));
    return((x >> 16) | (x << 16));

}

uint32_t prbset_to_bitmask() {
  uint32_t mask=0;
  int nb = (int) ceilf((float) cell.nof_prb / srslte_ra_type0_P(cell.nof_prb));
  for (int i=0;i<nb;i++) {
    if (i >= prbset_orig && i < prbset_orig + prbset_num) {
      mask = mask | (0x1<<i);
    }
  }
  return reverse(mask)>>(32-nb);
}

int update_radl(uint32_t sf_idx) {

  bzero(&ra_dl, sizeof(srslte_ra_dl_dci_t));
  ra_dl.harq_process = 0;
  ra_dl.mcs_idx = mcs_idx;
  ra_dl.ndi = 0;
  ra_dl.rv_idx = 0;
  ra_dl.alloc_type = SRSLTE_RA_ALLOC_TYPE0;
  ra_dl.type0_alloc.rbg_bitmask = prbset_to_bitmask();

  srslte_ra_pdsch_fprint(stdout, &ra_dl, cell.nof_prb);
  srslte_ra_dl_grant_t dummy_grant;
  srslte_ra_dl_dci_to_grant(&ra_dl, &dummy_grant, cell, sf_idx, cfi, true);
  srslte_ra_dl_grant_fprint(stdout, &dummy_grant);
  printf("Type new MCS index and press Enter: "); fflush(stdout);

  if (dummy_grant.mcs.tbs > dummy_grant.nof_bits) {
    fprintf(stderr, "Invalid code rate %d/%d=%.2f\n", dummy_grant.mcs.tbs,
dummy_grant.nof_bits, (float) dummy_grant.mcs.tbs / dummy_grant.nof_bits);
    return -1;
  }

  return 0;
}

#define DATA_BUFF_SZ    100000
uint8_t data[8*DATA_BUFF_SZ], data_unpacked[DATA_BUFF_SZ];
uint8_t data_tmp[DATA_BUFF_SZ];

uint8_t

float calcul_varianza(float snr, cf_t *subFrame_samples, int sf_samples){
        float varianza,So,suma,sumaTotal=0.0;
        int i;


        for(i=0;i<sf_samples;i++){
                suma=   __real__    subFrame_samples[i]    *    __real__
```

```
subFrame_samples[i] + __imag__ subFrame_samples[i] *__imag__
subFrame_samples[i];
                sumaTotal=suma+sumaTotal;
                ;
        }
        So=sumaTotal/(float)sf_samples;

        varianza= (float) sqrt((double)So/pow(10.0,(double) snr/10.0));

        return varianza;
}

int interpolar(cf_t *input, cf_t *output, uint32_t input_samples){

        int i, j=0, k;
   uint32_t interp=15360/input_samples;
   for(i=0;i<sf_n_samples;i++){
          for(k=0;k<interp;k++){
        output[j]=input[i];
        j++;
          }
   }
   return j;
}

int main(int argc, char **argv) {
  int nf=0, sf_idx=0, N_id_2=0;
  cf_t pss_signal[SRSLTE_PSS_LEN];
  float sss_signal0[SRSLTE_SSS_LEN]; // for subframe 0
  float sss_signal5[SRSLTE_SSS_LEN]; // for subframe 5
  uint8_t bch_payload[SRSLTE_BCH_PAYLOAD_LEN];
  int i;
  cf_t *sf_symbols[SRSLTE_MAX_PORTS];
  cf_t *slot1_symbols[SRSLTE_MAX_PORTS];
  srslte_dci_msg_t dci_msg;
  srslte_dci_location_t locations[SRSLTE_NSUBFRAMES_X_FRAME][30];
  uint32_t sfn;
  srslte_chest_dl_t est;
        struct timeval tdata[3];
        int exec_time;


  float varianza;

  cf_t output_buffer_inter[15360];
  bzero(output_buffer_inter,15360*sizeof(cf_t));

#ifdef DISABLE_UHD
  if (argc < 3) {
    usage(argv[0]);
    exit(-1);
  }
#endif

  parse_args(argc, argv);
```

```
  //N_id_2 per la creació de la PSS associada
  N_id_2 = cell.id % 3;
  //Número de RE en cada subframe
  sf_n_re = 2 * SRSLTE_CP_NORM_NSYMB * cell.nof_prb * SRSLTE_NRE;
  //Número de mostres en temporals en un subframe
  sf_n_samples = 2 * SRSLTE_SLOT_LEN(srslte_symbol_sz(cell.nof_prb));

  cell.phich_length = SRSLTE_PHICH_NORM;
  cell.phich_resources = SRSLTE_PHICH_R_1;
  sfn = 0;

  prbset_num = (int) ceilf((float) cell.nof_prb / srslte_ra_type0_P(cell.nof_prb));
  last_prbset_num = prbset_num;

  /* inicialització del canals i de la ifft */
  base_init();

  /* Generate PSS/SSS signals */
  srslte_pss_generate(pss_signal, N_id_2);
  srslte_sss_generate(sss_signal0, sss_signal5, cell.id);

  /* Generate CRS signals */
  if (srslte_chest_dl_init(&est, cell)) {
    fprintf(stderr, "Error initializing equalizer\n");
    exit(-1);
  }

  /* Initiate valid DCI locations */
  for (i=0;i<SRSLTE_NSUBFRAMES_X_FRAME;i++) {
    srslte_pdcch_ue_locations(&pdcch, locations[i], 50, i, cfi, 1234);

  }

  for (i = 0; i < SRSLTE_MAX_PORTS; i++) { // now there's only 1 port
    sf_symbols[i] = sf_buffer;
    slot1_symbols[i] = &sf_buffer[SRSLTE_SLOT_LEN_RE(cell.nof_prb, cell.cp)];
  }

//Configurem la USRP per rebre dades.
#ifndef DISABLE_UHD
  if (!output_file_name) {
    printf("Set TX rate: %.2f MHz\n",
        cuhd_set_tx_srate(uhd, srslte_sampling_freq_hz(cell.nof_prb)) / 1000000);
    printf("Set TX gain: %.1f dB\n", cuhd_set_tx_gain(uhd, uhd_gain));
    printf("Set TX freq: %.2f MHz\n",
        cuhd_set_tx_freq(uhd, uhd_freq) / 1000000);
  }
#endif

  if (update_radl(sf_idx)) {
    exit(-1);
  }

  nf = 0;
```

```
 bool send_data = false;

 mcs_idx_system=mcs_idx;

 while (nf < nof_frames || nof_frames == -1) {
   for (sf_idx = 0; sf_idx < SRSLTE_NSUBFRAMES_X_FRAME && (nf < nof_frames
|| nof_frames == -1); sf_idx++) {
        //Símbols del subframe
        bzero(sf_buffer, sizeof(cf_t) * sf_n_re);

        //Introduim les senyals generades PSS i SSS en el subframe
        //quan aquest sigui el 0 i el 5
        if (sf_idx == 0 || sf_idx == 5) {
                srslte_pss_put_slot(pss_signal, sf_buffer, cell.nof_prb,
SRSLTE_CP_NORM);
                srslte_sss_put_slot(sf_idx ? sss_signal5 : sss_signal0, sf_buffer,
cell.nof_prb,
                                SRSLTE_CP_NORM);
        }
        //Introduim les senyals pilots al subframe.
        srslte_refsignal_cs_put_sf(cell, 0, est.csr_signal.pilots[0][sf_idx], sf_buffer);
        //Agrupem la informació del MIB en bits i la introduim en la variable
bch_payload
        srslte_pbch_mib_pack(&cell, sfn, bch_payload);
        //Codifiquem el canal PBCH amb els bits del MIB, i ho fem solament en el
subframe 0
        if (sf_idx == 0) {
                srslte_pbch_encode(&pbch, bch_payload, slot1_symbols);
        }
        //Codifiquem el canal PCFICH amb el cfi previament inicialitzat (CFI=2)
        srslte_pcfich_encode(&pcfich, cfi, sf_symbols, sf_idx);

                //Enviem bits aleatoris
                  for(i=0;i<pdsch_cfg.grant.mcs.tbs;i++) {
                          data[i] = rand()%2];
                  }

       //En aquesta funció introduim els parametres del DCI al PDCCH
       srslte_ra_dl_dci_to_grant(&ra_dl, &pdsch_cfg.grant, cell, sf_idx, cfi, true);
       //En paquetem la informació del DCI i el número de RB que ha de decodificar el
UE
       srslte_dci_msg_pack_pdsch(&ra_dl, &dci_msg, SRSLTE_DCI_FORMAT1,
cell.nof_prb, false);
       //Codifiquem el PDCCH amb les localitzacion pels UE previament fetes i
       //i amb la ID del RNTI per tal que pugui codificar be el bloc del CRC,
       //tot això codificat amb el número de símbols indicats en el cfi
       if (srslte_pdcch_encode(&pdcch, &dci_msg, locations[sf_idx][0], 1234,
sf_symbols, sf_idx, cfi)) {
         fprintf(stderr, "Error encoding DCI message\n");
         exit(-1);
       }
       //Observem si tenim segmentació amb el TBS associat a la transmissió
       srslte_cbsegm(&pdsch_cfg.cb_segm, pdsch_cfg.grant.mcs.tbs);
       srslte_softbuffer_tx_reset(&softbuffer);
```

```
        pdsch_cfg.sf_idx = sf_idx;
        pdsch_cfg.rv = 0;
        //Codifiquem les dades dintre de la variable 'data'
        if (srslte_pdsch_encode(&pdsch, &pdsch_cfg, &softbuffer, data, sf_symbols)) {
          fprintf(stderr, "Error encoding PDSCH\n");
          exit(-1);
        }

      /* Transform to OFDM symbols */
      srslte_ofdm_tx_sf(&ifft, sf_buffer, output_buffer);

      /* send to file or usrp */
      if (output_file_name) {
        if (!null_file_sink) {
          varianza=calcul_varianza(SNR,sf_buffer,sf_n_re);
          srslte_ch_awgn_c(output_buffer,output_buffer,varianza,sf_n_samples);
          //INTERPOLAR ¡¡ESTO FUERA EN la app real!!
          int output_samples=0;
          output_samples=interpolar(output_buffer,output_buffer_inter,sf_n_samples);

          srslte_filesink_write(&fsink, output_buffer, sf_n_samples);
          srslte_filesink_write(&fsink2, output_buffer_inter, output_samples);
        }
        usleep(1000);
      } else {
#ifndef DISABLE_UHD
        // FIXME
        float norm_factor = (float) cell.nof_prb/15/sqrtf(pdsch_cfg.grant.nof_prb);
        srslte_vec_sc_prod_cfc(output_buffer,  uhd_amp*norm_factor,  output_buffer,
SRSLTE_SF_LEN_PRB(cell.nof_prb));
        cuhd_send(uhd, output_buffer, sf_n_samples, true);
#endif
      }

      nf++;
    }
    sfn = (sfn + 1) % 1024;
  }

  base_free();

  printf("Done\n");
  exit(0);
}
```

ANNEX 3: CODI DE LA IMPLEMENTACIÓ DE L'UE

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <sys/time.h>

#include "srslte/srslte.h"

char *input_file_name;
char *output_file_name="aux.txt";
int nof_frames=100, frame_length=1920, symbol_sz=128;

#define CFO_AUTO  -9999.0
float force_cfo = CFO_AUTO;

typedef enum {FIND=0,TRACK=1};

//printf colors
#define BLACK "\E[m"
#define RED "\E[31m"
#define GREEN "\E[32m"
#define YELLOW "\E[33m"
#define BLUE "\E[34m"
#define MAGENTA "\E[35m"
#define CYAN "\E[36m"
#define NEGRITA "\E[1m"
#define SUBRAYADO "\E[4m"

//CIRCULAR BUFFER
typedef struct buffer_control{

        uint32_t writeIndex;
        uint32_t readIndex;
        uint32_t buffsize;
        uint32_t occuplevel;

}buffctrl;

float BLER=0.0;
float ERROR=0.0 ;
int frame_decode=0;

void usage(char *prog) {
  printf("Usage: %s [olntsNfcv] -i input_file\n", prog);
  printf("\t-o output_file [Default %s]\n", output_file_name);
  printf("\t-l frame_length [Default %d]\n", frame_length);
  printf("\t-n number of frames [Default %d]\n", nof_frames);
  printf("\t-v srslte_verbose\n");
}

void parse_args(int argc, char **argv) {
```

```
  int opt;
  while ((opt = getopt(argc, argv, "ionltsNfcv")) != -1) {
    switch(opt) {
    case 'i':
      input_file_name = argv[optind];
      break;
    case 'n':
      nof_frames = atoi(argv[optind]);
      break;
    case 'l':
      frame_length = atoi(argv[optind]);
      break;
    default:
      usage(argv[0]);
      exit(-1);
    }
  }
  if (!input_file_name) {
    usage(argv[0]);
    exit(-1);
  }
}



/**
* void write(buffer* buffer, _Complex float value, int length)
* writes value into the buffer
* @param buffer* buffer
*   pointer to buffer to be used
* @param _Complex float value
*   value to be written in buffer
*/

int writeCbuff(buffctrl* buffer, _Complex float *buffdata, _Complex float *in, int length){
        int i;
        if(buffer->writeIndex >= buffer->readIndex){
                buffer->occuplevel = buffer->writeIndex - buffer->readIndex;
        }
        else{
                buffer->occuplevel=buffer->buffsize-(buffer->readIndex-buffer-
>writeIndex);
        }

        if(buffer->buffsize - buffer->occuplevel >= length){
                for(i=0; i<length; i++){
                        buffdata[buffer->writeIndex]=*(in+i);
                        buffer->writeIndex++;
                        if(buffer->writeIndex==buffer->buffsize){
                                buffer->writeIndex=0;
                        }
                }
                if(buffer->writeIndex >= buffer->readIndex){
                        buffer->occuplevel = buffer->writeIndex - buffer->readIndex;
                }
```

```
		else{
			buffer->occuplevel=buffer->buffsize-(buffer->readIndex-buffer->writeIndex);
		}

		return 0;
	}else{
		printf("Error writeCbuff: Not enough space in buffer\n");
		return -1;
	}
}
/**
* void readn(buffer* buffer, int Xn)
* reads specified value from buffer
* @param buffer* buffer
*   pointer to buffer to be read from
* @param int Xn
*   specifies the value to be read from buffer counting backwards from the most recently written value
*     i.e. the most recently writen value can be read with readn(buffer, 0), the value written before that with readn(buffer, 1)
*/
int readCbuff(buffctrl* buffer,_Complex float *buffdata, _Complex float *out, int length){
	int i;

	if(buffer->writeIndex >= buffer->readIndex){
		buffer->occuplevel = buffer->writeIndex - buffer->readIndex;
	}
	else{
		buffer->occuplevel=buffer->buffsize-(buffer->readIndex-buffer->writeIndex);
	}
	if(buffer->occuplevel >= length){
		for(i=0; i<length; i++){
			*(out+i) = buffdata[buffer->readIndex];
			buffer->readIndex++;
			if(buffer->readIndex==buffer->buffsize){
				buffer->readIndex=0;
			}
		}
		if(buffer->writeIndex >= buffer->readIndex){
			buffer->occuplevel = buffer->writeIndex - buffer->readIndex;
		}
		else{
			buffer->occuplevel=buffer->buffsize-(buffer->readIndex-buffer->writeIndex);
		}

		return 0;
	}else{
		printf("Error readCbuff: Not enough data in buffer\n");
		return -1;
	}
}
```

```
void setPSS_LPrealfilter128(float *LPfilter){

        LPfilter[0] =     -0.015417502473732522;
        LPfilter[1] =     0.003839956798763609;
        LPfilter[2] =     0.003719542523464047;
        LPfilter[3] =     0.003775421872687917;
        LPfilter[4] =     0.003938069472808964;
        LPfilter[5] =     0.004139142625385333;
        LPfilter[6] =     0.004313927309211304;
        LPfilter[7] =     0.004404210012805018;
        LPfilter[8] =     0.004361535922739865;
        LPfilter[9] =     0.004148775356547758;
        LPfilter[10] =    0.003742281920465906;
        LPfilter[11] =    0.003134672767019641;
        LPfilter[12] =    0.002336044095532337;
        LPfilter[13] =    0.001372878944504369;
        LPfilter[14] =    0.000286562936710045;
        LPfilter[15] =    -0.000867313244156237;
        LPfilter[16] =    -0.002021376538223761;
        LPfilter[17] =    -0.003104238319011994;
        LPfilter[18] =    -0.004041084721152157;
        LPfilter[19] =    -0.004759578119797244;
        LPfilter[20] =    -0.005197839065604633;
        LPfilter[21] =    -0.005305226770862375;
        LPfilter[22] =    -0.005051424445998523;
        LPfilter[23] =    -0.004425220671653666;
        LPfilter[24] =    -0.003440812557980871;
        LPfilter[25] =    -0.002137084622954199;
        LPfilter[26] =    -0.000576449853169101;
        LPfilter[27] =    0.001156506089656297;
        LPfilter[28] =    0.002958496174815677;
        LPfilter[29] =    0.004713461012609000;
        LPfilter[30] =    0.006299439147760158;
        LPfilter[31] =    0.007595418110620833;
        LPfilter[32] =    0.008488910497444458;
        LPfilter[33] =    0.008884794075845229;
        LPfilter[34] =    0.008713021525410697;
        LPfilter[35] =    0.007930526967643525;
        LPfilter[36] =    0.006540964541132228;
        LPfilter[37] =    0.004560356883938307;
        LPfilter[38] =    0.002087279041766787;
        LPfilter[39] =    -0.000780789841152774;
        LPfilter[40] =    -0.003895316862414899;
        LPfilter[41] =    -0.007070209098484606;
        LPfilter[42] =    -0.010104641381020277;
        LPfilter[43] =    -0.012789440357831189;
        LPfilter[44] =    -0.014911755891048536;
        LPfilter[45] =    -0.016265108615309280;
        LPfilter[46] =    -0.016664305000451397;
        LPfilter[47] =    -0.015959109740866215;
        LPfilter[48] =    -0.014042986295891345;
        LPfilter[49] =    -0.010858963401436746;
        LPfilter[50] =    -0.006405803316760139;
        LPfilter[51] =    -0.000742970747281631;
        LPfilter[52] =    0.006010241308861676;
```

```
LPfilter[53] =  0.013682086956477055;
LPfilter[54] =  0.022048009849418754;
LPfilter[55] =  0.030837083317143171;
LPfilter[56] =  0.039756264628880697;
LPfilter[57] =  0.048485786013107594;
LPfilter[58] =  0.056708256279830144;
LPfilter[59] =  0.064113531344801827;
LPfilter[60] =  0.070418693533264881;
LPfilter[61] =  0.075378939401439027;
LPfilter[62] =  0.078799065692138001;
LPfilter[63] =  0.080543668617574143;
LPfilter[64] =  0.080543668617574143;
LPfilter[65] =  0.078799065692138001;
LPfilter[66] =  0.075378939401439027;
LPfilter[67] =  0.070418693533264881;
LPfilter[68] =  0.064113531344801827;
LPfilter[69] =  0.056708256279830144;
LPfilter[70] =  0.048485786013107594;
LPfilter[71] =  0.039756264628880697;
LPfilter[72] =  0.030837083317143171;
LPfilter[73] =  0.022048009849418754;
LPfilter[74] =  0.013682086956477055;
LPfilter[75] =  0.006010241308861676;
LPfilter[76] =  -0.000742970747281631;
LPfilter[77] =  -0.006405803316760139;
LPfilter[78] =  -0.010858963401436746;
LPfilter[79] =  -0.010858963401436746;
LPfilter[80] =  -0.015959109740866215;
LPfilter[81] =  -0.016664305000451397;
LPfilter[82] =  -0.016265108615309280;
LPfilter[83] =  -0.014911755891048536;
LPfilter[84] =  -0.012789440357831189;
LPfilter[85] =  -0.010104641381020277;
LPfilter[86] =  -0.007070209098484606;
LPfilter[87] =  -0.003895316862414899;
LPfilter[88] =  -0.000780789841152774;
LPfilter[89] =  0.002087279041766787;
LPfilter[90] =  0.004560356883938307;
LPfilter[91] =  0.006540964541132228;
LPfilter[92] =  0.007930526967643525;
LPfilter[93] =  0.008713021525410697;
LPfilter[94] =  0.008884794075845229;
LPfilter[95] =  0.008488910497444458;
LPfilter[96] =  0.007595418110620833;
LPfilter[97] =  0.006299439147760158;
LPfilter[98] =  0.004713461012609000;
LPfilter[99] =  0.002958496174815677;
LPfilter[100] =  0.001156506089656297;
LPfilter[101] =  -0.000576449853169101;
LPfilter[102] =  -0.002137084622954199;
LPfilter[103] =  -0.003440812557980871;
LPfilter[104] =  -0.004425220671653666;
LPfilter[105] =  -0.005051424445998523;
LPfilter[106] =  -0.005305226770862375;
LPfilter[107] =  -0.005197839065604633;
```

```
        LPfilter[108] =  -0.004759578119797244;
        LPfilter[109] =  -0.004041084721152157;
        LPfilter[110] =  -0.003104238319011994;
        LPfilter[111] =  -0.002021376538223761;
        LPfilter[112] =  -0.000867313244156237;
        LPfilter[113] =  0.000286562936710045;
        LPfilter[114] =  0.001372878944504369;
        LPfilter[115] =  0.002336044095532337;
        LPfilter[116] =  0.003134672767019641;
        LPfilter[117] =  0.003742281920465906;
        LPfilter[118] =  0.004148775356547758;
        LPfilter[119] =  0.004361535922739865;
        LPfilter[120] =  0.004404210012805018;
        LPfilter[121] =  0.004313927309211304;
        LPfilter[122] =  0.004139142625385333;
        LPfilter[123] =  0.003938069472808964;
        LPfilter[124] =  0.003775421872687917;
        LPfilter[125] =  0.003719542523464047;
        LPfilter[126] =  0.003839956798763609;
        LPfilter[127] =  -0.015417502473732522;

}

int filtro(cf_t *input, cf_t *output, uint32_t input_length, uint32_t diezmado, float *filter,
uint32_t filter_length){

        int a;
        int i;
        int conv_output_len_real,conv_output_len_imag;
        float input_real[input_length+filter_length+1];
        float input_imag[input_length+filter_length+1];
        float conv_output_real[19200*2];
        float conv_output_imag[19200*2];
        bzero(&input_real[input_length],filter_length*sizeof(float));
        bzero(&input_imag[input_length],filter_length*sizeof(float));

        if (input_length >= filter_length) {
          setPSS_LPrealfilter128(filter);
                for(i=0;i<input_length;i++){
                        input_real[i]=__real__ input[i];
                        input_imag[i]=__imag__ input[i];
                }
          conv_output_len_real = srslte_conv_ff_dec(input_real, filter, conv_output_real,
input_length, filter_length,diezmado);

          conv_output_len_imag = srslte_conv_ff_dec(input_imag, filter,
conv_output_imag, input_length, filter_length,diezmado);

        } else {
          for (int i=0;i<15360;i++) {
                  a = srslte_vec_dot_prod_fff(filter, input_real, filter_length);
          }
          conv_output_len_real = input_length/diezmado;
          conv_output_len_imag = input_length/diezmado;
        }
```

```
        for(i=0;i<input_length;i++){
                __real__ output[i] = conv_output_real[i];
                __imag__ output[i] = conv_output_imag[i];
        }

        return conv_output_len_real;
}

/* CP detection algorithm taken from:
 * "SSS Detection Method for Initial Cell Search in 3GPP LTE FDD/TDD Dual Mode Receiver"
 * by Jung-In Kim et al.
 */
srslte_cp_t detect_cp(cf_t *input, uint32_t peak_pos, uint32_t fft_size)
{
        float M_norm_avg= 0.0,M_ext_avg=0.0;
  float R_norm, R_ext, C_norm, C_ext;
  float M_norm=0, M_ext=0;

  uint32_t cp_norm_len = SRSLTE_CP_LEN_NORM(7, fft_size);
  uint32_t cp_ext_len = SRSLTE_CP_LEN_EXT(fft_size);

  cf_t *input_cp_norm = &input[peak_pos-2*(fft_size+cp_norm_len)];
  cf_t *input_cp_ext = &input[peak_pos-2*(fft_size+cp_ext_len)];

  for (int i=0;i<2;i++) {
    R_norm          =      crealf(srslte_vec_dot_prod_conj_ccc(&input_cp_norm[fft_size],
input_cp_norm, cp_norm_len));
    C_norm  = cp_norm_len * srslte_vec_avg_power_cf(input_cp_norm, cp_norm_len);
    input_cp_norm += fft_size+cp_norm_len;
    M_norm += R_norm/C_norm;
  }

  M_norm_avg = SRSLTE_VEC_EMA(M_norm/2, M_norm_avg, 0.2);

  for (int i=0;i<2;i++) {
    R_ext   = crealf(srslte_vec_dot_prod_conj_ccc(&input_cp_ext[fft_size], input_cp_ext,
cp_ext_len));
    C_ext  = cp_ext_len * srslte_vec_avg_power_cf(input_cp_ext, cp_ext_len);
    input_cp_ext += fft_size+cp_ext_len;
    if (C_ext > 0) {
      M_ext += R_ext/C_ext;
    }
  }

  M_ext_avg = SRSLTE_VEC_EMA(M_ext/2, M_ext_avg, 0.2);

  if (M_norm_avg > M_ext_avg) {
    return SRSLTE_CP_NORM;
  } else if (M_norm_avg < M_ext_avg) {
    return SRSLTE_CP_EXT;
  } else {
    if (R_norm > R_ext) {
      return SRSLTE_CP_NORM;
```

```
    } else {
      return SRSLTE_CP_EXT;
    }
  }
}

int delmar(cf_t *input, cf_t *output, uint32_t input_length, uint32_t output_length){

        int delmar=input_length/output_length;
        int i,j=0;

        for(i=0;i<input_length;i=i+delmar){
                output[j]=input[i];
                j++;
        }
        return j;
}

int init_cfo(srslte_cfo_t cfo[4]){

        int i,j;

        uint32_t length[4]={1920,3840,7680,15360};

        for(i=0;i<=3;i++){
                if (srslte_cfo_init(&cfo[i], length[i])) {
                        fprintf(stderr, "Error initiating CFO\n");
                        return -1;
                }
        }
        return 1;
}

int init_PSS_SSS(srslte_pss_synch_t pss[3][4][2], srslte_sss_synch_t sss[3][4]){

        int i,j;
        uint32_t length[4]={1920,3840,7680,15360};
        uint32_t symbol_size[4]={128,256,512,1024};

        for(i=0;i<=2;i++){
                for(j=0;j<=3;j++){

                        if (srslte_pss_synch_init_fft(&pss[i][j][FIND], length[j],
symbol_size[j])) {
                          fprintf(stderr, "Error initializing PSS object [%d][%d][FIND]\n",i,j);
                          exit(-1);
                        }
                        if (srslte_pss_synch_set_N_id_2(&pss[i][j][FIND], i)) {
                          fprintf(stderr, "Error initializing N_id_2 [%d][%d][FIND]\n",i,j);
                          exit(-1);
                        }

                        if (srslte_pss_synch_init_fft(&pss[i][j][TRACK], symbol_size[j]+62,
symbol_size[j])) {
                          fprintf(stderr, "Error initializing PSS object [%d][%d][FIND]\n",i,j);
```

```
                        exit(-1);
                    }
                    if (srslte_pss_synch_set_N_id_2(&pss[i][j][TRACK], i)) {
                      fprintf(stderr, "Error initializing N_id_2 [%d][%d][FIND]\n",i,j);
                      exit(-1);
                    }

                    if (srslte_sss_synch_init(&sss[i][j], symbol_size[j])) {
                      fprintf(stderr, "Error initializing SSS object [%d][%d][FIND]\n",i,j);
                      exit(-1);
                    }
                    if (srslte_sss_synch_set_N_id_2(&sss[i][j], i)) {
                      fprintf(stderr, "Error initializing N_id_2 [%d][%d][FIND]\n",i,j);
                      exit(-1);
                    }

                }
        }
        return 1;
}

int free_PSS_SSS(srslte_pss_synch_t pss[3][4][2], srslte_sss_synch_t sss[3][4]){

        int i,j;

        for(i=0;i<=2;i++){
                for(j=0;j<4;j++){
                        srslte_pss_synch_free(&pss[i][j][FIND]);
                        srslte_pss_synch_free(&pss[i][j][TRACK]);
                        srslte_sss_synch_free(&sss[i][j]);
                }
        }
        return 1;
}

int find_index_pss_sss(uint32_t symbol_size){
        if(symbol_size==128) return 0;
        else if(symbol_size==256) return 1;
        else if(symbol_size==512) return 2;
        else if(symbol_size==1024) return 3;
        else return -1;
}

int init_buffer_circular(buffctrl *buffCtrl,uint32_t length_buffCtrl, cf_t *bufferC, uint32_t
length_bufferC){
        bzero(bufferC, sizeof(cf_t)*length_bufferC);
        buffCtrl->writeIndex = 0;
        buffCtrl->readIndex = 0;
        buffCtrl->buffsize = length_buffCtrl;
        buffCtrl->occuplevel=0;

        return 1;
}

int decode_SSS(cf_t *aux, srslte_pss_synch_t *pss, srslte_sss_synch_t *sss, uint32_t
```

```
N_id_2,float *cfo, int frame_cnt, uint32_t peak_pos,float peak_value, uint32_t
symbol_sz, srslte_cell_t *cell){
        uint32_t m0, m1;
        float m0_value, m1_value;
        uint32_t sf_idx;
        uint32_t cell_id;
        uint32_t sss_idx;

        cell->cp=detect_cp(aux,peak_pos,symbol_sz);
        sss_idx                                    =                                    peak_pos-
2*(symbol_sz+SRSLTE_CP_LEN(symbol_sz,(SRSLTE_CP_NSYMB(cell-
>cp)==7)?SRSLTE_CP_NORM_LEN:SRSLTE_CP_LEN_EXT(symbol_sz)));

        if(sss_idx >= 0){
                srslte_sss_synch_m0m1_diff(sss, &aux[sss_idx], &m0, &m0_value,
&m1, &m1_value);
        sf_idx=srslte_sss_synch_subframe(m0, m1);
        cfo[frame_cnt] = srslte_pss_synch_cfo_compute(pss, &aux[peak_pos-
symbol_sz]);
        cell_id=3*srslte_sss_synch_N_id_1(sss,m0,m1)+N_id_2;
        cell->id=cell_id;
        /*printf("\n\tFr.Cnt\tN_id_2\tN_id_1\tSubf\tPSS
Peak/Avg\tIdx\tm0\tm1\tCFO\tcell\n");
        printf("\t==========================================================
============================\n");
        printf("\t%d\t%d\t%d\t%d\t%.3f\t\t%3d\t%d\t%d\t%.3f\t",
                        frame_cnt,N_id_2, srslte_sss_synch_N_id_1(sss, m0, m1),
                        sf_idx, peak_value,
                        peak_pos, m0, m1,
                        cfo[frame_cnt]); printf("%d\n",cell->id);*/
        }
        return sf_idx;
}

int decode_PBCH(cf_t *aux, srslte_ue_mib_t *ue_mib, srslte_cell_t *cell, uint32_t *sfn){

        int n;
        uint8_t bch_payload[SRSLTE_BCH_PAYLOAD_LEN];
        uint32_t SFN;
        uint32_t sfn_offset;

        srslte_pbch_decode_reset(&ue_mib->pbch);
        n = srslte_ue_mib_decode(ue_mib, aux, bch_payload, &cell->nof_ports,
&sfn_offset);
        if (n < 0) {
                fprintf(stderr, "Error decoding UE MIB\n");
        }else if (n == SRSLTE_UE_MIB_FOUND) {
                srslte_pbch_mib_unpack(bch_payload, cell, &SFN);
                //srslte_cell_fprint(stdout, cell, SFN);
                //printf("Decoded MIB. SFN: %d, offset: %d\n", SFN, sfn_offset);
                SFN = (SFN + sfn_offset)%1024;
                sfn=SFN;
                //printf("GENERAL. SFN: %d\n",sfn);
        }
```

```
	return n;
}

int init_UE_dl_funtion(srslte_ue_dl_t *ue_dl,srslte_cell_t cell, uint16_t rnti){
	 if (srslte_ue_dl_init(ue_dl, cell)) {  // This is the User RNTI
		fprintf(stderr, "Error initiating UE downlink processing module\n");
		exit(-1);
	 }

	 // Configure downlink receiver for the SI-RNTI since will be the only one we'll
use
	 srslte_ue_dl_set_rnti(ue_dl, 1234);
return 1;
}

int decode_UE_dl(cf_t *aux, srslte_ue_dl_t *ue_dl, uint16_t rnti, uint8_t *data, uint32_t
sf_idx, uint32_t sfn){

	int n;

	if (1234 != SRSLTE_SIRNTI) {
		n = srslte_ue_dl_decode(ue_dl, aux, data, sf_idx);
	} else {
		n = srslte_ue_dl_decode_rnti_rv(ue_dl, aux, data, sf_idx,
                                                                SRSLTE_SIRNTI,
((int) ceilf((float)3*(((sfn)/2)%4)/2))%4);
	}

	printf("\n\t n=%d",n);
	return n;
}

int aliniar( buffctrl *buffer, int pos_last_read, uint32_t peak_pos, uint32_t
perfect_peak_pos, bool diezmado){
	if(diezmado==true)		buffer->readIndex=pos_last_read+peak_pos-
perfect_peak_pos;
	else				buffer->readIndex=pos_last_read+(peak_pos-
perfect_peak_pos)*(frame_length/(perfect_peak_pos*2));
	if((int)buffer->readIndex >= (int)buffer->buffsize){
		buffer->readIndex=buffer->readIndex-buffer->buffsize;
	}
	else if((int)buffer->readIndex < 0) {
		buffer->readIndex=buffer->readIndex+buffer->buffsize;

	}

	return 1;
}

int main(int argc, char **argv) {
	srslte_filesource_t fsrc;
	srslte_filesink_t fsink;
	int frame_cnt;
	  //SYNC PSS y SSS
	int pss_pos_symb_sz;
```

```
srslte_pss_synch_t pss[3][4][2]; // One for each N_id_2
srslte_sss_synch_t sss[3][4]; // One for each N_id_2
srslte_cfo_t cfocorr[4];
int peak_pos[3];
int peak_pos_128;
float cfo[19200];
float peak_value[3];
uint32_t m0, m1;
float m0_value, m1_value;
uint32_t N_id_2;
uint32_t N_id_2_128;
uint32_t sf_idx;
float corr_peak_threshold;
int sss_idx;

//SYSTEM
srslte_cell_t cell;
int conx_loss_cont;
int cont_correlacio;

//DECODE INFO

/* TODO: Do something with the output data */
uint8_t data_packed[20000];
srslte_ue_dl_t ue_dl;


//circular buffer
buffctrl buffCtrl;
buffctrl buffCtrl128;
int pos_last_read;
int pos_last_read_128;

int bufferroom;

int numsubframes;

int n;
int kk;

/* read all file or nof_frames */
bool init_ue_dl;
bool init_ue_mib;
bool SYNCRO_OK;
bool sync_PSS_SSS;
bool sync_PSS_SSS_128;
bool sync_MIB;
bool sync_MIB_128;
bool alinear_128;
bool alinear;
bool fft_tracking;
bool fft_tracking_init;
bool filtre_sf;
//MIB
uint8_t bch_payload[SRSLTE_BCH_PAYLOAD_LEN];
```

```
srslte_ue_mib_t ue_mib;
uint32_t sfn;
uint32_t sfn_offset;


struct timeval tdata[3];
int *exec_time;

if (argc < 3) {
usage(argv[0]);
exit(-1);
}

parse_args(argc,argv);


cf_t input[frame_length];
cf_t aux[10*frame_length];
cf_t aux_2[10*frame_length];
int frame_aux;

cf_t bufferC[25*frame_length];
cf_t bufferC128[10*frame_length];

float filter[128];

gettimeofday(&tdata[1], NULL);
printf("Initializing...");fflush(stdout);

if (srslte_filesource_init(&fsrc, input_file_name, SRSLTE_COMPLEX_FLOAT)) {
fprintf(stderr, "Error opening file %s\n", input_file_name);
exit(-1);
}
if (srslte_filesink_init(&fsink, output_file_name, SRSLTE_COMPLEX_FLOAT)) {
fprintf(stderr, "Error opening file %s\n", output_file_name);
exit(-1);
}

exec_time = malloc(nof_frames*sizeof(int));
if (!exec_time) {
perror("malloc");
exit(-1);
}


init_cfo(cfocorr);

/* We have 2 options here:
* a) We create 3 pss objects, each initialized with a different N_id_2
* b) We create 1 pss object which scans for each N_id_2 one after another.
* a) requries more memory but has less latency and is paralellizable.
*/


init_PSS_SSS(pss,sss);
```

```
	pss_pos_symb_sz=0;


	init_buffer_circular(&buffCtrl,frame_length*30,bufferC,frame_length*25);
	init_buffer_circular(&buffCtrl128,frame_length*20,bufferC128,frame_length*10);

	bzero(aux_2, sizeof(cf_t)*(frame_length));

	N_id_2=-1;
	N_id_2_128=0;
	sf_idx=-1;
	sss_idx=0;
	sfn=0;
	init_ue_dl=false;
	init_ue_mib=false;
	SYNCRO_OK=false;
	sync_PSS_SSS=false;
	sync_PSS_SSS_128=false;
	sync_MIB=false;
	sync_MIB_128=false;
	alinear_128=false;
	alinear=false;
	fft_tracking_init=false;
	fft_tracking=false;
	filtre_sf=true;
	corr_peak_threshold=2.0;

	conx_loss_cont=0;
	cont_correlacio=0;
	frame_aux=0;

	peak_pos[0]=0;
	peak_pos[1]=0;
	peak_pos[2]=0;
	peak_pos_128=0;

	gettimeofday(&tdata[2], NULL);
	get_time_interval(tdata);
	printf("done in %d s %d ms\n", (int) tdata[0].tv_sec, (int) tdata[0].tv_usec);

	/* read all file or nof_frames */
	frame_cnt = 0;
	while (frame_length == srslte_filesource_read(&fsrc, input, frame_length)
	  && frame_cnt < nof_frames) {

	//Verificamos si hay suficientes posiciones disponibles en el buffer
	bufferroom=buffCtrl.buffsize-buffCtrl.occuplevel;

	if(frame_length>bufferroom){
		printf("El número de datos recividos es superior a la que puede
introducirse en el bufferCircular. length_frame %d > bufferoom %d",
frame_length,bufferroom);
		frame_cnt=nof_frames;
	}
```

```
writeCbuff(&buffCtrl,bufferC,input,frame_length);

//Number of available subframes
numsubframes = (buffCtrl.occuplevel/frame_length);

if(numsubframes>=1){

pos_last_read_128=buffCtrl128.readIndex;
pos_last_read=buffCtrl.readIndex;

readCbuff(&buffCtrl, bufferC, aux, frame_length);

//DELMAR PER EXTREURE EL SENYAL
if(sync_MIB_128==true){
        delmar(aux, aux, frame_length, frame_aux);
        pss_pos_symb_sz=find_index_pss_sss(symbol_sz);
}
//s-APLICA UN FILTRE/DELMADOR PER EXTREURE LES SUBPORT CENTRALS PER SYNCRO Y EXTREURE EL MIB
        if((sync_PSS_SSS_128==false || sync_MIB_128== false) && filtre_sf==true){
                frame_aux=filtro(aux,aux,frame_length,8,filter,128);
                writeCbuff(&buffCtrl128,bufferC128,aux,frame_aux);
                readCbuff(&buffCtrl128, bufferC128, aux, frame_aux);
                pss_pos_symb_sz=find_index_pss_sss(symbol_sz);
    }


        //srslte_filesink_write(&fsink, aux, frame_aux);
        if(sf_idx==-1 || sf_idx==0 || sf_idx==5){


                if (force_cfo != CFO_AUTO) {
                        srslte_cfo_correct(&cfocorr[pss_pos_symb_sz], aux, aux, force_cfo/symbol_sz);
                }

                if((int)N_id_2==-1){
                        N_id_2=N_id_2_128;

                                peak_pos[N_id_2]= srslte_pss_synch_find_pss(&pss[N_id_2][pss_pos_symb_sz][FIND], aux, &peak_value[N_id_2]);

                                if(cont_correlacio==4){
                                        cont_correlacio=0;
                                        N_id_2_128++;
                                        if(N_id_2_128 >= 3 ){
                                                N_id_2_128=0;
                                        }
                                }
                }else{
                        if(fft_tracking==false){
                                peak_pos[N_id_2]=
```

```
srslte_pss_synch_find_pss(&pss[N_id_2][pss_pos_symb_sz][TRACK],
&aux[peak_pos_128*frame_aux/1920-symbol_sz-10], &peak_value[N_id_2]);

        peak_pos[N_id_2]=peak_pos[N_id_2]+peak_pos_128*frame_aux/1920-
symbol_sz-10;
                                }else{
                                                peak_pos[N_id_2]=
srslte_pss_synch_find_pss(&pss[N_id_2][pss_pos_symb_sz][TRACK],
&aux[frame_aux/2-symbol_sz-10], &peak_value[N_id_2]);

        peak_pos[N_id_2]=peak_pos[N_id_2]+frame_aux/2-symbol_sz-10;
                                        }
                                }

                                if (peak_value[N_id_2] > corr_peak_threshold) {
                                        conx_loss_cont=0;

                                        if(peak_pos[N_id_2]!=960 && frame_aux == 1920
&& sync_PSS_SSS_128 == false) {

                                                //ALINIAMOS EL BUFFER CIRCULAR 128
PARA QUE PUEDA DECODIFICAR BIEN EL MIB
                                                aliniar( &buffCtrl128, pos_last_read_128,
peak_pos[N_id_2], 960,true);
                                                alinear_128=true;
                                                sf_idx--;
                                                fft_tracking_init=false;
                                                peak_pos_128=peak_pos[N_id_2];

                                        }else if(peak_pos[N_id_2]!=frame_aux/2 &&
(frame_aux != 1920 || sync_PSS_SSS_128 == true)) {

                                                aliniar( &buffCtrl, pos_last_read,
peak_pos[N_id_2], frame_aux/2, false);
                                                alinear=true;
                                                sf_idx--;
                                                if(sf_idx==-1) sf_idx=9;

                                        }else if(peak_pos[N_id_2]==960 &&
sync_PSS_SSS_128 == false){
                                                //Extraccio de la SSS quan ens
sincronitzem les primeres vegades a 1.4MHz
                                                if(fft_tracking_init==false)
                                                        sf_idx=decode_SSS(aux,
&pss[N_id_2][pss_pos_symb_sz][FIND], &sss[N_id_2][pss_pos_symb_sz],N_id_2,cfo,
frame_cnt, peak_pos[N_id_2],peak_value[N_id_2], symbol_sz, &cell);
                                                else
                                                        sf_idx=decode_SSS(aux,
&pss[N_id_2][pss_pos_symb_sz][TRACK],
&sss[N_id_2][pss_pos_symb_sz],N_id_2,cfo, frame_cnt,
peak_pos[N_id_2],peak_value[N_id_2], symbol_sz, &cell);


                                                if(sf_idx==0){
                                                        sync_MIB_128=true;
```

```
                                        sync_PSS_SSS_128=true;
                                    }

                                    pos_last_read=buffCtrl.readIndex;
                                }else   if(peak_pos[N_id_2]==frame_aux/2    &&
sync_PSS_SSS_128 == true){
                                    //Extraccio de la SSS quan estem totalment
sincronitzats
                                    if(fft_tracking==false)
                                        sf_idx=decode_SSS(aux,
&pss[N_id_2][pss_pos_symb_sz][FIND],  &sss[N_id_2][pss_pos_symb_sz],N_id_2,cfo,
frame_cnt, peak_pos[N_id_2],peak_value[N_id_2], symbol_sz, &cell);
                                    else
                                        sf_idx=decode_SSS(aux,
&pss[N_id_2][pss_pos_symb_sz][TRACK],
&sss[N_id_2][pss_pos_symb_sz],N_id_2,cfo,                frame_cnt,
peak_pos[N_id_2],peak_value[N_id_2], symbol_sz, &cell);

                                    sync_MIB=true;
                                    sync_PSS_SSS=true;

                                    pos_last_read=buffCtrl.readIndex;
                                }
                                if(fft_tracking_init==false){
                                    //Tracking de la PSS
                                    fft_tracking_init=true;
                                    fft_tracking=true;
                                }

                            }else{
                                //Aquesta part de codi es per quan intentem
sincronitzar i el pic no sobrepassa el TH

                                cont_correlacio++;
                                if(sync_PSS_SSS_128==false){
                                    N_id_2=-1;
                                }
                                conx_loss_cont++;
                                if(conx_loss_cont==2){

                                    buffCtrl.readIndex=buffCtrl.readIndex-3000;
                                    if((int)buffCtrl.readIndex < 0)
        buffCtrl.readIndex=buffCtrl.readIndex+buffCtrl.buffsize;


                                    sync_PSS_SSS=false;
                                    sync_MIB=false;
                                    sync_MIB_128=false;
                                    sync_PSS_SSS_128=false;
                                    init_ue_mib=false;
                                    SYNCRO_OK=false;
                                    alinear=false;
                                    alinear_128=false;
                                    numsubframes=0;
```

```
                                        if(sync_PSS_SSS==true){
                                                buffCtrl.occuplevel=0;

        init_buffer_circular(&buffCtrl128,frame_length*20,bufferC128,frame_length*10);
                                                frame_aux=1920;
                                                symbol_sz=128;
                                        }

                                        N_id_2=-1;
                                        sf_idx=-1;
                                        conx_loss_cont=0;
                                }
                        }


        srslte_pss_synch_reset(&pss[0][pss_pos_symb_sz][FIND]);

        srslte_pss_synch_reset(&pss[1][pss_pos_symb_sz][FIND]);

        srslte_pss_synch_reset(&pss[2][pss_pos_symb_sz][FIND]);

        srslte_pss_synch_reset(&pss[0][pss_pos_symb_sz][TRACK]);

        srslte_pss_synch_reset(&pss[1][pss_pos_symb_sz][TRACK]);

        srslte_pss_synch_reset(&pss[2][pss_pos_symb_sz][TRACK]);

                        if(sf_idx==0 && sync_MIB_128==true && sync_MIB==false){
                                for(kk=0;kk<frame_aux;kk++) aux_2[kk]=aux[kk];
                                filtre_sf=false;
                        }

                }
                if(sf_idx==1 && sync_MIB_128==true && sync_MIB==false && filtre_sf==false){

                        cell.nof_prb=SRSLTE_UE_MIB_NOF_PRB;
                        cell.nof_ports=0;

                         if (srslte_ue_mib_init(&ue_mib, cell)) {
                                fprintf(stderr, "Error initaiting UE MIB decoder\n");
                                exit(-1);
                         }

                         n = decode_PBCH(aux_2, &ue_mib, &cell, &sfn);

                        if(n==0){
                                 //En aaquest moment el senyal s-ha sincronitzat en BW de 1.4MHz i pot descodificar el MIB a 1.4Mhz
                                 sync_MIB_128=false;
                                 sync_PSS_SSS_128=false;
                        }else if (n == SRSLTE_UE_MIB_FOUND) {

                                        if(alinear_128==true)   sf_idx++;
```

```
                                        sync_PSS_SSS_128=true;
                                        //INICIALIZACIÓN PARA syncronizar el subframe
real TX
                                        frame_aux=2                                    *
SRSLTE_SLOT_LEN(srslte_symbol_sz(cell.nof_prb));
                                        symbol_sz=srslte_symbol_sz(cell.nof_prb);

                                        fft_tracking_init=false;
                                        fft_tracking=false;

                                        init_ue_mib=true;
                        }

        }else if(sf_idx==0 && sync_MIB==true && sync_PSS_SSS==true){

                n = decode_PBCH(aux, &ue_mib, &cell, &sfn);

                if(n==SRSLTE_UE_MIB_FOUND){
                        SYNCRO_OK=true;
                        if(init_ue_dl==false){

                                init_UE_dl_funtion(&ue_dl,cell,1234);

                                init_ue_dl=true;

                        }
                }

                }else if(sf_idx==3 && init_ue_mib==true){

                        srslte_ue_mib_free(&ue_mib);
                        if (srslte_ue_mib_init(&ue_mib, cell)) {
                                fprintf(stderr, "Error initaiting UE MIB decoder\n");
                                exit(-1);
                        }
                        init_ue_mib=false;

                }

                if(SYNCRO_OK==true ){
                        frame_decode++;

                        n=decode_UE_dl(aux,&ue_dl,1234,data_packed, sf_idx, sfn);

                        if (n < 0) {
                         fprintf(stderr, "Error decoding UE DL\n");fflush(stdout);
                        } else if (n > 0) {
                                printf("\n\t DECODE PDSCH OK\n\n");
                        } else if(n==0){
                                ERROR++;
                                printf("\n\t NOT DECODING ANYTHING\n\n");
                        }

                }
```

```
    }

    if(sf_idx!=-1){
                    sf_idx++;
          }
    if (sf_idx == 10) {
                    sf_idx=0;
          }

    if(frame_decode>100) BLER=ERROR/((float)frame_decode);
    printf("\nERROR=%f , frame_decode=%f -->
BLER=%f",ERROR,(float)frame_decode,BLER);

    frame_cnt++;
  }


  srslte_filesource_free(&fsrc);
  srslte_filesink_free(&fsink);
  free_PSS_SSS(pss,sss);

  printf("Done\n");
  exit(0);
}
```